Grado en Ingeniería Electrónica, Industrial y Automática

TRABAJO FIN DE GRADO

Aceleración hardware de funciones trigonométricas

Autor: Rubén Sevilla Serrano

Tutora: Celia López Ongil

Octubre de 2013

Título: Aceleración hardware de funciones trigonométricas
Autor: Rubén Sevilla Serrano
Tutor: Celia López Ongil

TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y la lectura del Trabajo Fin de Grado el día 4 de Octubre de 2013 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerdan otorgarle la calificación de

PRESIDENTE

VOCAL SECRETARIO

Agradecimientos

En primer lugar querría mostrar mi más sincero agradecimiento a la tutora del proyecto, Celia López Ongil, por su cordial colaboración y disponibilidad siempre que lo he necesitado, y sobre todo, por sus ayudas y consejos que me han servido durante estos meses para seguir forjando mi futuro.

Agradecer a mis padres y hermanos el apoyo mostrado durante toda la carrera, tanto a nivel personal como económico, ya que sin ellos esto no hubiera sido posible, de corazón, gracias.

Finalmente, dar las gracias a todos mis amigos que han estado presentes durante estos cuatro años, y que por supuesto, espero que lo estén durante toda mi vida.

A todos ellos, gracias.

La más estricta justicia no creo que sea siempre la mejor política.
Abraham Lincoln

Resumen

En la actualidad hay una gran tendencia hacia el desarrollo de circuitos electrónicos digitales en FPGAs y sistemas embebidos. Esto se debe principalmente a las ventajas que estos dispositivos ofrecen frente a otras soluciones más sofisticadas. Este trabajo se centra en los dispositivos reconfigurables denominados FPGAs. Estos son la solución a numerosas aplicaciones, siempre y cuando tengan los recursos necesarios. La adopción de FPGAS en la industria ha sido impulsada debido a que estos dispositivos combinan lo mejor de los Circuitos Integrados de Aplicación Específica (ASICs) y de los sistemas basados en microprocesadores. Las grandes ventajas que presentan las FPGAs son las siguientes: son reprogramables (lo que añade una enorme flexibilidad al flujo de diseño), los costes de desarrollo y adquisición para pequeñas cantidades tienen un bajo coste comparado con los ASICs y además; el tiempo de desarrollo también es menor que en estos últimos.

Este trabajo Fin de Grado consiste en el diseño de un módulo digital que realiza funciones trigonométricas de forma eficiente en términos de velocidad de procesamiento y en utilización de recursos, independientemente de la tecnología escogida para su implementación final, pero aprovechando los recursos existentes actualmente en los dispositivos presentes en el mercado. Se han aplicado técnicas de aceleración hardware y se han diseño distintas arquitecturas para evaluar su eficacia en la optimización de parámetros como velocidad, recursos, hardware, etc. También hemos utilizado distintas precisiones en cada diseño, las cuales tendrán su implicación en los diferentes resultados.

Finalmente, se ha realizado una implementación hardware del bloque, como prueba de su viabilidad, en dispositivos reconfigurables (FPGAs) del fabricante Xilinx. No obstante, el módulo diseñado es independiente de la tecnología final y puede implementarse en cualquier FPGA o ASIC.

La utilización del bloque diseñado e implementado es muy interesante para numerosas aplicaciones que realizan tareas de procesamiento de datos para control, cálculo o comunicaciones, un ejemplo es el control digital de sistemas de potencia [1].

Índice general

CAPÍTULOS. Apartados.

1. INTRODUCCIÓN 1
1.1. Justificación 1
1.2. Objetivos 2
1.3. Etapas del trabajo 2
1.4. Estructura del documento 3
1.5. Medios utilizados 4
2. DESCRIPCIÓN GENERAL DE LA APLICACIÓN 6
2.1. Fundamento teórico 6
2.2. Cálculo de seno y coseno 8
2.3. Métodos aplicados para el cálculo del seno 10
2.3.1. Aproximación por muestreo 10
2.3.2. Interpolación lineal 11
2.3.3. Serie de Taylor 14
3. METODOLOGÍAS DEL DISEÑO HARDWARE 17
3.1. Herramientas CAD 17
3.2. Metodología de diseño *top-down* 18
3.3. Descripción de diseño 18
4. LENGUAJE DE DESCRIPCIÓN DE HARDWARE VHDL 20
4.1. Introducción 20
4.2. Descripción 20
4.2.1. Comportamiento 21
4.2.2. Estructura 21
4.3. Conceptos básicos 22
4.3.1. Entidades y arquitecturas 22
4.3.2. Diseño estructural 23
4.3.3. Objetos 24
4.3.4. Sentencias concurrentes y secuenciales 26
4.3.5. Diseño genérico 29
4.3.6. Paquetes 30
4.3.7. Bibliotecas 30
4.4. Tipos de datos y operadores 31
4.4.1. Enteros 31

4.4.2. Reales ... 31
4.4.3. Físicos ... 31
4.4.4. Enumerados ... 32
4.4.5. Vectores ... 33
4.4.6. Operadores ... 34
4.4.7. Atributos ... 34
4.5. Estructuras combinacionales ... 35
4.5.1. Sentencias condicionales ... 35
4.5.2. Sentencias condicionales secuenciales ... 36
4.5.3. Sentencias condicionales concurrentes ... 37
4.5.4. Reglas para el diseño de circuitos combinacionales ... 37
4.6. Estructuras secuenciales ... 38
4.6.1. Circuitos síncronos y asíncronos ... 38
4.6.2. Biestables y registros ... 39
4.6.3. Reglas para el diseño de procesos secuenciales ... 39
4.6.4. Máquinas de estados finitos ... 40
4.7. Operaciones iterativas ... 42
4.8. Simulación ... 43
4.8.1. Bancos de pruebas ... 43
4.8.2. Ficheros ... 44
5. DISEÑO DIGITAL DE LA FUNCIÓN SENO ... 45
5.1. Herramienta de desarrollo ... 45
5.1.1. Xilinx ISE Design Suite 13.2 ... 45
5.2. Formato de los datos ... 46
5.2.1. Longitud de datos almacenados ... 46
5.2.2. Formato de datos almacenados ... 47
5.2.3. Tablas para datos de entrada ... 48
5.3. Diagramas de flujo ... 50
5.3.1. Diagramas para la aproximación e interpolación ... 51
5.3.2. Diagrama para la serie de Taylor ... 53
5.4. El seno mediante aproximación por muestreo e interpolación lineal ... 53
5.4.1. Introducción ... 53
5.4.2. Longitudes de 8 y 16 bits ... 54
5.4.3. Diseño del diseño con 8 y 16 bits ... 55
5.4.4. Longitud de 32 bits ... 58
5.4.5. Diseño con 32 bits ... 58
5.5. El seno mediante la serie de Taylor ... 61
5.5.1. Operaciones en formato decimal ... 62
5.5.2. Segmentación ... 63
5.5.3. Diseño ... 66
5.5.4. Función *aproximar* ... 71
5.5.5. Función *suma* ... 73
6. SIMULACIÓN, VALIDACIÓN E IMPLEMENTACIÓN ... 76
6.1. Herramientas utilizadas ... 76
6.1.1. Simulador digital ISim 0.61xd ... 76
6.1.2. Software matemático MATLAB R2011a ... 77
6.2. Valores de entrada para las simulaciones ... 77
6.3. Simulaciones y cálculo de errores ... 78
6.3.1. Cálculo del seno mediante aproximación por muestreo ... 78
6.3.2. Cálculo del seno mediante interpolación lineal ... 84

6.3.3. Cálculo del seno mediante la serie de Taylor 90
6.3.4. Resumen del cálculo de errores 91
6.4. Implementación en FPGA 92
6.4.1. FPGA Spartan-3E Starter Kit Board 92
6.4.2. Diagrama de bloques 95
6.4.3. Diseño del bloque contador 95
6.4.4. Diseño del bloque seno 96
6.4.5. Función *fdisplay_ang* 106
6.4.6. Función *fdisplay_sen* 107
6.5. Análisis de tiempos 108
7. CONCLUSIONES 109
7.1. Acerca del uso de VHDL 109
7.2. Acerca de la exactitud de los resultados 110
7.3. Líneas de trabajo futuras 110
8. PRESUPUESTO 111
8.1. Tareas 111
8.2. Recursos 112
8.3. Resumen de costes 112
ANEXOS 113
Código de los diseños en VHDL 113
Bancos de pruebas de los diseños 128
Simulaciones de los diseños 134
Código en MATLAB para el cálculo de errores 138
GLOSARIO 142
BIBLIOGRAFÍA 143

Índice de figuras

CAPÍTULOS. Figuras.

1. INTRODUCCIÓN

2. DESCRIPCIÓN GENERAL DE LA APLICACIÓN

2.1. Triángulo rectángulo 7
2.2. Circunferencia goniométrica 8
2.3. Cálculos mediante la circunferencia goniométrica 8
2.4. Relaciones trigonométricas más significativas 9
2.5. Gráfica de función seno y coseno 10
2.6. Seno aproximado mediante 13 muestras 10
2.7. Seno aproximado mediante 25 muestras 11
2.8. Seno aproximado mediante 37 muestras 11
2.9. Ejemplo de interpolación 12
2.10. Teorema de Tales 12
2.11. Seno interpolado mediante 13 muestras 13
2.12. Seno interpolado mediante 25 muestras 14
2.13. Seno interpolado mediante 37 muestras 14
2.14. Resultados de la serie de Taylor para distinto nº de términos 16
2.15. Seno mediante serie de Taylor con seis términos 16

3. METODOLOGÍAS DEL DISEÑO HARDWARE

3.1. Esquema descripción comportamental 19
3.2. Esquema descripción estructural 19
3.3. Esquema descripción RTL 19

4. LENGUAJE DE DESCRIPCIÓN DE HARDWARE VHDL

4.1. Estructura de una entidad 21
4.2. Estructura de una entidad con componentes 21
4.3. Ejemplo de declaración de una entidad 22
4.4. Ejemplo de declaración de una arquitectura 22
4.5. Esquema sumador de 1 bit 23
4.6. Código sumador de 1 bit 23
4.7. Esquema sumador de 2 bits 24
4.8. Código sumador de 2 bits 25
4.9. Ejemplo del uso de señales 26
4.10. Consideraciones en el uso de señales 26

4.11. Sentencias concurrentes 27
4.12. Sentencias secuenciales 27
4.13. Ejemplo de función suma 28
4.14. Ejemplo del procedimiento suma 28
4.15. Ejemplo del diseño genérico en un contador de N-bits 29
4.16. Instanciación de componente con parámetro genérico 29
4.17. Paquete 30
4.18. Sentencias condicionales 36
4.19. Sentencia IF 36
4.20. Sentencia CASE 36
4.21. Asignación condicional 37
4.22. Asignación seleccionada 37
4.23. Biestable D con reset 39
4.24. Modelo general para procesos secuenciales 39
4.25. Modelo de Huffman para máquina de estados finitos 40
4.26. Ejemplo máquina de estados de Moore 40
4.27. Máquina de Moore en VHDL 41
4.28. Ejemplo máquina de estados de Mealy 41
4.29. Máquina de Mealy en VHDL 42
4.30. Descripción en VHDL de FOR y WHILE 42
4.31. Descripción en VHDL de NEXT y EXIT 43
4.32. Estructura de banco de pruebas 43
4.33. Ficheros 44

5. DISEÑO DIGITAL DE LA FUNCIÓN SENO

5.1. Diagrama del funcionamiento por aproximación e interpolación (8 y 16 bits) 51
5.2. Diagrama del funcionamiento por aproximación e interpolación (32 bits) 52
5.3. Diagrama del funcionamiento mediante la serie de Taylor 53
5.4. División con bucle FOR 55
5.5. Arquitectura del diseño por aproximación e interpolación para 8 y 16 bits 55
5.6. Ruta de datos del diseño por aproximación e interpolación para 8 y 16 bits 56
5.7. Bibliotecas, paquetes y entidad. Diseño aproximación e interpolación (8 bits) 56
5.8. Arquitectura en VHDL. Diseño aproximación e interpolación (8 bits) 57
5.9. Acceso a tabla y aproximación. Diseño aproximación e interpolación (8 bits) 57
5.10. Interpolación. Diseño aproximación e interpolación (8 bits) 58
5.11. Arquitectura del diseño por la aproximación e interpolación para 32 bits 59
5.12. Ruta de datos del diseño por aproximación e interpolación para 32 bits 59
5.13. Bibliotecas, paquetes y entidad. Diseño aproximación e interpolación (32 bits) ... 60
5.14. Arquitectura en VHDL. Diseño aproximación e interpolación (32 bits) 60
5.15. Acceso a tabla y aproximación. Diseño aproximación e interpolación (32 bits) ... 61
5.16. Interpolación. Diseño aproximación e interpolación (32 bits) 61
5.17. Suma binaria 62
5.18. Multiplicación binaria I 62
5.19. Multiplicación binaria II 62
5.20. Diseño 1 con segmentación 63
5.21. Cronograma del diseño 1 63
5.22. Diseño 2 con segmentación 64
5.23. Cronograma del diseño 2 64
5.24. Segmentación del diseño de la función seno mediante la Serie de Taylor 65
5.25. Cronograma del diseño de la función seno mediante la Serie de Taylor 65
5.26. Arquitectura del diseño mediante la Serie de Taylor 66

5.27. Ruta de datos del diseño mediante la Serie de Taylor 67
5.28. Bibliotecas, paquetes y entidad. Serie de Taylor 67
5.29. Arquitectura en VHDL y señales creadas. Serie de Taylor 68
5.30. Estudio de cuadrante y primer registro. Serie de Taylor 68
5.31. Segundo y tercer registro para retrasar señales. Serie de Taylor 69
5.32. Cálculo de x^2 y x^3. Serie de Taylor 69
5.33. División de x^3 y x^5. Serie de Taylor 70
5.34. Suma de términos y resultado. Serie de Taylor 71
5.35. Función aproximar 72
5.36. Visualización función aproximar 72
5.37. Función suma 75
6. SIMULACIÓN, VALIDACIÓN E IMPLEMENTACIÓN
6.1. Resultados del diseño por aproximación con 13 muestras (8 bits) 79
6.2. Resultados del diseño por aproximación con 25 muestras (8 bits) 79
6.3. Resultados del diseño por aproximación con 37 muestras (8 bits) 80
6.4. Resultados del diseño por aproximación con 13 muestras (16 bits) 81
6.5. Resultados del diseño por aproximación con 25 muestras (16 bits) 81
6.6. Resultados del diseño por aproximación con 37 muestras (16 bits) 82
6.7. Resultados del diseño por aproximación con 13 muestras (32 bits) 83
6.8. Resultados del diseño por aproximación con 25 muestras (32 bits) 83
6.9. Resultados del diseño por aproximación con 37 muestras (32 bits) 84
6.10. Resultados del diseño por interpolación con 13 muestras (8 bits) 85
6.11. Resultados del diseño por interpolación con 25 muestras (8 bits) 86
6.12. Resultados del diseño por interpolación con 37 muestras (8 bits) 86
6.13. Resultados del diseño por interpolación con 13 muestras (16 bits) 87
6.14. Resultados del diseño por interpolación con 25 muestras (16 bits) 87
6.15. Resultados del diseño por interpolación con 37 muestras (16 bits) 88
6.16. Resultados del diseño por interpolación con 13 muestras (32 bits) 89
6.17. Resultados del diseño por interpolación con 25 muestras (32 bits) 89
6.18. Resultados del diseño por interpolación con 37 muestras (32 bits) 90
6.19. Resultados del diseño mediante la serie de Taylor 91
6.20. Spartan-3E XC3S500E 93
6.21. Placa adaptación para pines 93
6.22. Displays individual (SA52-11EWA) y cuádruple (LM5644R) de 7 segmentos 94
6.23. Montaje final Spartan-3E XC3S500E 94
6.24. Diagrama de bloques de la implementación 95
6.25. Entidad contador 95
6.26. Arquitectura de la entidad contador 96
6.27. Entidad seno 96
6.28. Arquitectura de la entidad seno 97
6.29. Rebotes 98
6.30. Conformador de pulsos 98
6.31. Anti-rebotes 98
6.32. Diseño VHDL del anti-rebotes 99
6.33. Máquina de estados de Moore simplificada 100
6.34. Diseño VHDL de la máquina de estados I 101
6.35. Diseño VHDL de la máquina de estados II 102
6.36. Diagrama de bloques del estado *ANG_DISPLAY* 103
6.37. Diseño VHDL de la máquina de estados III 104
6.38. Resultado del seno de 50º sin temporización y con temporización de retardo 105

6.39. Ejemplo display: seno 90º = 1 105
6.40. Ejemplo display: seno153º = 0.454 105
6.41. Ejemplo display: seno 357º = -0.052 105
6.42. Esquema visual 7 segmentos 106
6.43. Función *fdisplay_ang* 106
6.44. Función *fdisplay_sen* 107
7. CONCLUSIONES
8. PRESUPUESTO

Índice de tablas

CAPÍTULOS. Tablas.

1. INTRODUCCIÓN
2. DESCRIPCIÓN GENERAL DE LA APLICACIÓN
2.1. Relaciones trigonométricas 7
2.2. Relación de ángulos respecto el I cuadrante 9
3. METODOLOGÍAS DEL DISEÑO HARDWARE
4. LENGUAJE DE DESCRIPCIÓN DE HARDWARE VHDL
4.1. Valores del tipo STD_LOGIC 32
4.2. Principales operadores 34
4.3. Atributos de vector 35
5. DISEÑO DIGITAL DE LA FUNCIÓN SENO
5.1. Ejemplo conversión vector a decimal 48
5.2. Ángulos según el número de muestras 48
5.3. Tabla de ángulos y senos con 37 muestras (8 bits) 48
5.4. Tabla del seno con 37 muestras (16 bits) 49
5.5. Tabla del seno con 37 muestras (32 bits) 49
5.6. Correspondencia entre grados y radianes 50
5.7. Valores del dividendo x_2-x_1 54
5.8. Formato de almacenamiento decimal 73
5.9. Cálculo de errores en las divisiones de los términos 74
6. SIMULACIÓN, VALIDACIÓN E IMPLEMENTACIÓN
6.1. Resultados del diseño por aproximación con 8 bits 78
6.2. Resultados del diseño por aproximación con 16 bits 80
6.3. Resultados del diseño por aproximación con 32 bits 82
6.4. Resultados del diseño por interpolación con 8 bits 85
6.5. Resultados del diseño por interpolación con 16 bits 87
6.6. Resultados del diseño por interpolación con 32 bits 88
6.7. Resultados del diseño mediante la serie de Taylor 90
6.8. Resumen de errores calculados 92
6.9. Resumen de tiempos 108
7. CONCLUSIONES
8. PRESUPUESTO
8.1. Resumen del presupuesto 112

Capítulo 1

Introducción

En este primer capítulo se detallan la motivación del Trabajo Fin de Grado, los objetivos de este, y se hace una descripción de aspectos tales como las etapas del trabajo, la estructura y los medios utilizados para su realización.

1.1. Justificación

En la actualidad se pueden destacar dos tendencias claras en la generación de sistemas digitales, que son el desarrollo mediante sistemas embebidos (dispositivos que incluyen microprocesador junto a lógica dedicada) o mediante dispositivos configurables (FPGAs, *Field Programmable Gate Array)*.

El primer caso consiste en utilizar un módulo de computación ya diseñado para realizar una o varias tareas específicas, junto con una serie de periféricos que se encargan de conectar el módulo con el exterior. Sus componentes son un microprocesador, distintos tipos de memorias según la aplicación (una memoria de datos y/o de programa, memoria caché, memoria para almacenamiento masivo de datos), periféricos para comunicación serie/paralelo, para interfaces con usuarios, etc. Normalmente, estos sistemas se pueden programar con el lenguaje ensamblador del micro, sin embargo, también se podrán utilizar lenguajes de más alto nivel como C, C++ o Java. Esta solución permite realizar aplicaciones de forma sencilla (el diseñador no tiene que tener grandes conocimientos de electrónica digital), barata (hay una amplia oferta en el mercado) y rápida (un programa software personaliza de forma inmediata la aplicación).

El segundo caso consiste en utilizar dispositivos reconfigurables (FPGAs), que contienen bloques de lógica que pueden interconectarse y cuya funcionalidad puede

configurarse. Estos dispositivos pueden implementar funciones muy complejas, para lo que incluso contienen memorias y microprocesadores embebidos. En este caso se diseña la aplicación mediante un circuito digital a la medida y se usan lenguajes especializados, como son lenguajes de descripción hardware VHDL, Verilog y Abel. La lógica programable de estos dispositivos nos da la posibilidad de reproducir funciones tan sencillas como puertas lógicas y también, complejos procesados de señal para cálculo, control o comunicaciones. Las ventajas que presentan estos sistemas son que se pueden reconfigurar fácilmente, sus costes de adquisición y desarrollo son bajos así como el tiempo de desarrollo, en comparación con los sistemas microprocesador o con el diseño de un circuito digital a medida (ASIC). Históricamente, son la evolución de las PAL y los CPLD. Aunque el proceso de diseño requiere más conocimientos que la primera solución, el resultado es más eficiente y el coste suele ser menor.

La evolución en prestaciones que ha tenido la tecnología digital en los últimos años, permite resolver problemas complejos que mediante la tecnología analógica son inviables. Además, la existencia de potentes dispositivos reconfigurables en la actualidad nos proporciona flexibilidad a la hora de modificar nuestro diseño durante su fase de operación, con poco coste y en poco tiempo [2].

1.2. Objetivos

El objetivo principal del proyecto es lograr un diseño óptimo y eficaz de funciones trigonométricas de un módulo digital implementable en dispositivos hardware. Como ya se ha visto en el apartado anterior, debido a la tendencia actual del uso de FPGAs, se buscará diseñar el hardware a la medida para generar un seno mediante diferentes arquitecturas, buscando siempre la solución más eficiente.

Partiendo de este objetivo principal, surgen una serie de objetivos parciales:

- Estudio de los distintos métodos matemáticos para calcular la función seno.
- Definición de distintas arquitecturas digitales que implementen la función seno. Descripción de las mismas mediante el lenguaje de descripción de hardware VHDL.
- Simulación de las distintas arquitecturas. Obtención de resultados para su validación posterior.
- Cálculo de errores respecto a la herramienta matemática MATLAB.
- Validación de los resultados obtenidos.
- Síntesis del diseño para su implementación en FPGA.

1.3. Etapas del trabajo

En este apartado se establecen las fases en las que se ha dividido el proyecto. Este consta de tres fases principales, las cuales se explican a continuación:

- **Fase 1. Planificación.**

  - **Estudio de los métodos matemáticos disponibles:** Selección de las distintas soluciones para generar la función seno.

  - **Estudio de las tecnologías necesarias:** Estudio del lenguaje de descripción de hardware VHDL a utilizar, así como de la herramienta de simulación ISIM para analizar los resultados del diseño.

- **Fase 2. Diseño y desarrollo.**

  - **Diseño hardware de cada método.** Definición de la arquitectura, en nivel de transferencia de registros, que implementa cada método. Definición de cada bloque de la arquitectura, desde un punto de vista funcional y de su interfaz con el exterior. Descripción de los mismos en VHDL.

  - **Simulación.** Definición de los bancos de pruebas. Descripción y simulación.

  - **Síntesis.** Implementación del diseño más óptimo en tecnología FPGA.

  - **Evaluación.** Estudio de los resultados obtenidos mediante el cálculo de errores cometidos para cada método frente a una herramienta de cálculo matemático para PC (Matlab).

- **Fase 3. Documentación.**

  - **Memoria del Proyecto Fin de Carrera.** Redacción del presente documento.

  - **Presentación.** Desarrollo de la presentación audiovisual, la cual se usará en la defensa del proyecto ante el tribunal.

1.4. Estructura del documento

En este apartado se presenta un breve comentario de lo que trata cada capítulo de la presente memoria.

Capítulo 1. Introducción.

Se detalla el propósito y los objetivos del trabajo Fin de Grado. Se describen las fases del desarrollo, la estructura del documento y los medios empleados.

Capítulo 2. Descripción general de la aplicación.

Descripción del diseño realizado desde una perspectiva general. Detalle del fundamento teórico para cada método, desde un punto de vista matemático.

Capítulo 3. Metodología del diseño hardware.

Este capítulo describe el ciclo de diseño de un sistema hardware digital mediante el uso de lenguajes de descripción hardware y herramientas de síntesis lógica.

Capítulo 4. Lenguaje de descripción de hardware VHDL.

Contiene una descripción breve del lenguaje que se va a utilizar para el diseño digital. Este lenguaje es VHDL, y es el más utilizado para el desarrollo de diseño hardware. Se definen los elementos del lenguaje, como son los tipos de datos, operadores, operaciones, estructuras, etc.

Capítulo 5. Diseño digital de la función seno.

Aquí es donde se describe el diseño hardware de la función seno. Se describirán las arquitecturas y los bloques definidos para la implementación de los métodos seleccionados en el capítulo 2.

Capítulo 6. Simulación, verificación e implementación.

Descripción de las simulaciones realizadas para verificar el correcto funcionamiento de los diseños. Posteriormente, se estudian los resultados obtenidos mediante el cálculo de errores.

Capítulo 7. Conclusiones.

En este capítulo se realiza una descripción de las conclusiones extraídas en la realización del trabajo y se habla brevemente de las líneas de trabajo futuras.

Anexos: en este apartado se incluyen los códigos en VHDL usados para el diseño, bancos de pruebas, estudio de errores, etc.

Glosario: resumen de los acrónimos utilizados.

Bibliografía: donde se pueden encontrar las citas bibliográficas consultadas para la realización del proyecto y la memoria, webs, libros, etc.

1.5. Medios utilizados

Los medios empleados para la realización del Trabajo Fin de Grado han sido los siguientes:

Dispositivos Hardware:

- Ordenador portátil HP Pavilion g6-2104ss
- Spartan-3E Starter Kit Board XC3S500E

Aplicaciones software:

- Herramienta para el diseño de circuitos digitales y su prototipado, Xilinx ISE Design Suite 13.2.
- Simulador digital ISim 0.61xd.
- Software matemático MATLAB R2011a.
- Paquete Microsoft Office 2010.

Capítulo 2

Descripción general de la aplicación

La aplicación creada consiste en un módulo hardware digital que realiza la función seno. Se han realizado tres arquitecturas diferentes que realizan la misma función pero que atienden a distintos requisitos, como son la precisión, la velocidad de operación o el área necesaria para implementar el circuito digital. Todas las arquitecturas utilizan componentes básicos de las bibliotecas de los fabricantes de circuitos integrados, con el objeto de que los diseños presentados sean independientes de la tecnología. Los diseños pueden ser implementados en dispositivos reprogramables (FPGAs) o en circuitos integrados a la medida (ASICs), pero dadas las altas prestaciones de los primeros en la actualidad, parece más interesante su uso, tanto en coste como en tiempo de desarrollo. En el trabajo y para los diferentes métodos se ha llevado a cabo el estudio del error cometido frente a la función matemática exacta, el fundamento matemático en que se apoya, la velocidad de ejecución para el desarrollo de hardware, etc.

En el mundo actual, la trigonometría es una herramienta fundamental para arquitectos, ingenieros y otros profesionales. Puede hallar la altura de edificios, procesar imágenes, etc. Mientras que en ingeniería, las funciones trigonométricas se utilizan en numerosos algoritmos de procesamiento de datos, de control de procesos, etc.

2.1. Fundamento teórico

La trigonometría es una rama de las matemáticas, cuyo nombre proviene del griego y quiere decir “medición de triángulos”. Se basa en el estudio de las razones trigonométricas: seno, coseno, tangente; secante, cosecante y cotangente. Interviene en el resto de ramas de las matemáticas. Una vez conocida una de las razones se puede encontrar el resto, ya que existe relación entre todas ellas.

El cálculo de todas las razones trigonométricas se hace mediante el estudio de los lados y ángulos de un triángulo rectángulo. Con este estudio se obtienen todos los resultados y relaciones existentes entre las razones trigonométricas.

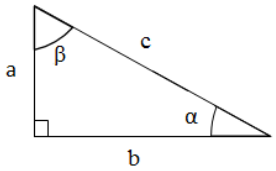


Figura 2.1. Triángulo rectángulo

Razones trigonométricas principales

$$sen(\alpha) = \frac{a}{c} = cos(\beta)$$

$$cos(\alpha) = \frac{b}{c} = sen(\beta)$$

$$tg(\alpha) = \frac{a}{b} = \frac{1}{tg(\beta)}$$

Razones trigonométricas secundarias

$$sec(\alpha) = \frac{1}{cos(\alpha)} = \frac{c}{b} = \frac{1}{sen(\beta)}$$

$$csc(\alpha) = \frac{1}{sen(\alpha)} = \frac{c}{a} = \frac{1}{cos(\beta)}$$

$$ctg(\alpha) = \frac{1}{tg(\alpha)} = \frac{b}{a} = tg(\beta)$$

Conociendo cualquiera de ellas se puede conocer el resto, debido a la relación directa que existe entre todas, la cual se muestra en el siguiente cuadro [3]:

	sen	**cos**	**tg**	**sec**	**csc**	**ctg**
$\boldsymbol{sen(\alpha)}$	$sen(\alpha)$	$\sqrt{1-cos^2(\alpha)}$	$\frac{tg(\alpha)}{\sqrt{1+tg^2(\alpha)}}$	$\frac{\sqrt{sec^2(\alpha)-1}}{sec(\alpha)}$	$\frac{1}{csc(\alpha)}$	$\frac{1}{\sqrt{1+ctg^2(\alpha)}}$
$\boldsymbol{cos(\alpha)}$	$\sqrt{1-sen^2(\alpha)}$	$cos(\alpha)$	$\frac{1}{\sqrt{1+tg^2(\alpha)}}$	$\frac{1}{sec(\alpha)}$	$\frac{\sqrt{csc^2(\alpha)-1}}{csc(\alpha)}$	$\frac{ctg(\alpha)}{\sqrt{1+ctg^2(\alpha)}}$
$\boldsymbol{tg(\alpha)}$	$\frac{sen(\alpha)}{\sqrt{1-sen^2(\alpha)}}$	$\frac{\sqrt{1-cos^2(\alpha)}}{cos(\alpha)}$	$tg(\alpha)$	$\sqrt{sec^2(\alpha)-1}$	$\frac{1}{\sqrt{csc^2(\alpha)-1}}$	$\frac{1}{ctg(\alpha)}$
$\boldsymbol{sec(\alpha)}$	$\frac{1}{\sqrt{1-sen^2(\alpha)}}$	$\frac{1}{cos(\alpha)}$	$\sqrt{1+tg^2(\alpha)}$	$sec(\alpha)$	$\frac{csc(\alpha)}{\sqrt{csc^2(\alpha)-1}}$	$\frac{\sqrt{1+ctg^2(\alpha)}}{ctg(\alpha)}$
$\boldsymbol{csc(\alpha)}$	$\frac{1}{sen(\alpha)}$	$\frac{1}{\sqrt{1-cos^2(\alpha)}}$	$\frac{\sqrt{1+tg^2(\alpha)}}{tg(\alpha)}$	$\frac{sec(\alpha)}{\sqrt{sec^2(\alpha)-1}}$	$csc(\alpha)$	$\sqrt{1+ctg^2(\alpha)}$
$\boldsymbol{ctg(\alpha)}$	$\frac{\sqrt{1-sen^2(\alpha)}}{sen(\alpha)}$	$\frac{cos(\alpha)}{\sqrt{1-cos^2(\alpha)}}$	$\frac{1}{tg(\alpha)}$	$\frac{1}{\sqrt{sec^2(\alpha)-1}}$	$\sqrt{csc^2(\alpha)-1}$	$ctg(\alpha)$

Tabla 2.1. Relaciones trigonométricas

2.2. Cálculo de seno y coseno

Como se ha visto en el apartado anterior, una vez se tiene el valor de una de las razones trigonométricas se puede calcular el resto. Se muestra el cálculo de seno y coseno ya que son los más significativos. Aunque es cierto que con calcular uno valdría para obtener el otro.

Partiendo de una circunferencia de radio unidad y cuyo centro de coordenadas sea el (0,0) (conocida como circunferencia goniométrica), se irán representando distintos ángulos y obteniendo así sus correspondientes valores. Sobre el eje X se calcula el valor del coseno, mientras que el del seno se hace sobre el eje Y.

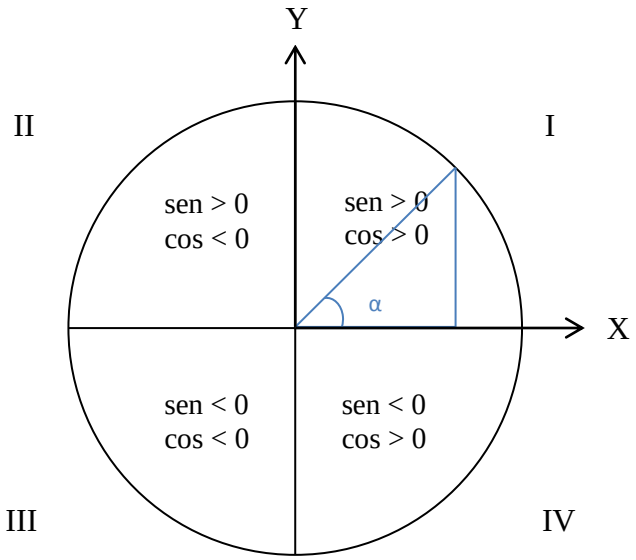


Figura 2.2. Circunferencia goniométrica

Como se ve en la circunferencia, se tienen cuatro cuadrantes. Estos cuadrantes se pueden relacionar todos con el primero de ellos, y a partir de ahí sacar el valor del seno y coseno, teniendo en cuenta el signo, debido a que en cada cuadrante el signo será diferente, tal y como se aprecia en la *figura 2.2*.

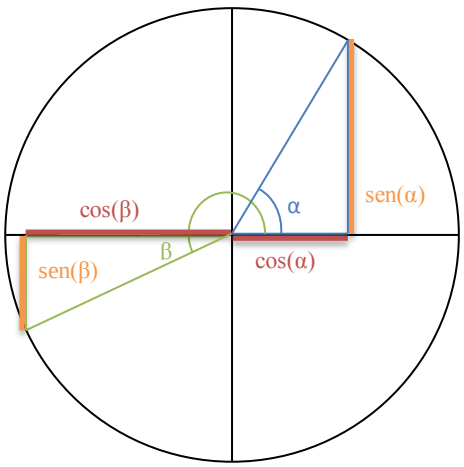


Figura 2.3. Cálculos mediante la circunferencia goniométrica

La relación que hay entre los ángulos de los distintos cuadrantes respecto con el primero, es la que se muestra en la *tabla 2.2*.

Cuadrante	Función	Relación respecto I cuadrante
II	$sen(\alpha)$	$sen(180-\alpha)$
	$cos(\alpha)$	$-cos(180-\alpha)$
III	$sen(\alpha)$	$-sen(\alpha-180)$
	$cos(\alpha)$	$-cos(\alpha-180)$
IV	$sen(\alpha)$	$-sen(360-\alpha)$
	$cos(\alpha)$	$cos(360-\alpha)$

Tabla 2.2. Relación de ángulos respectos al I cuadrante

Con un ejemplo se puede apreciar que daría igual calcular la función directamente o hacerlo con su relación correspondiente.

$$sen(235º) = -0.819 = -sen(235º - 180º) = -sen(55º) = -0.819$$

En la *figura 2.4* se representan las razones trigonométricas seno y coseno de los ángulos más significativos de la circunferencia goniométrica.

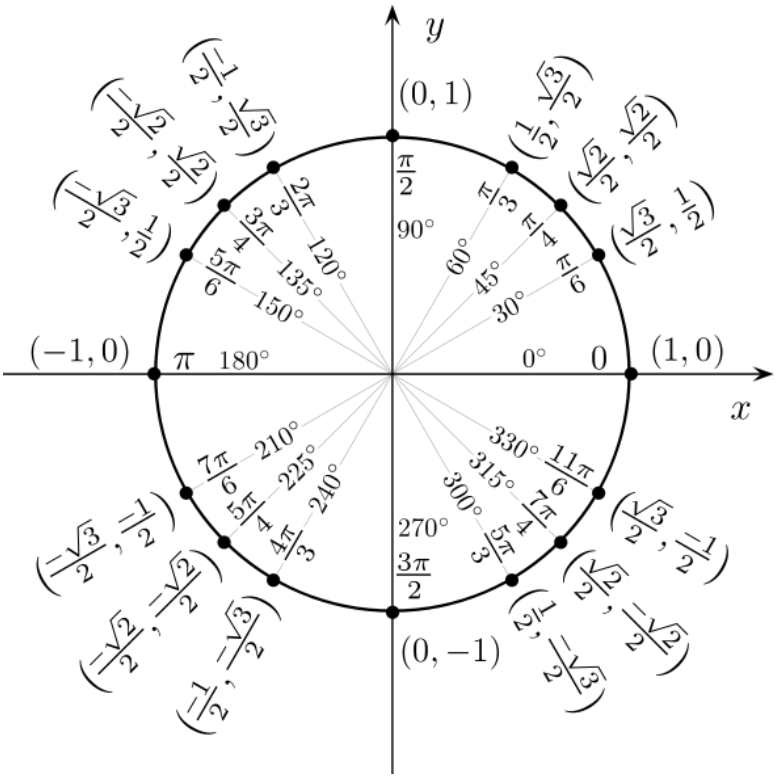


Figura 2.4. Relaciones trigonométricas más significativas [4]

La representación de las funciones seno y coseno en el infinito, es decir, si el número de ángulos a calcular fuese el intervalo (-∞,+∞), se tendría un comportamiento periódico, tal y como se puede ver en la siguiente figura.

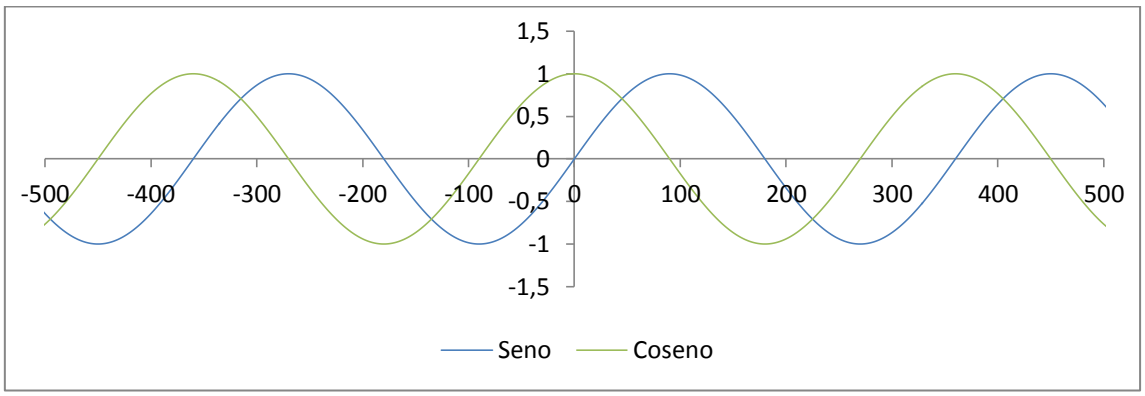


Figura 2.5. Gráfica función seno y coseno

2.3. Métodos aplicados para el cálculo del seno

En este apartado se explican los tres métodos que se han usado para calcular la función seno. Cada método dará un resultado aproximado, nunca exacto. El error que se genera en cada método será estudiado con mayor profundidad en el capítulo 6.

2.3.1. Aproximación por muestreo

En este método lo primero es muestrear la señal del seno, almacenando esos valores de muestra, a partir de los cuales se aproximan el resto. En este caso se ha muestreado el seno con tres relaciones de frecuencias diferentes respecto a la función seno, obteniendose 13, 25 o 37 datos. Es evidente que cuanto mayor sea el número de muestras, menor será el error, sin embargo, más largo será el proceso y más hardware se necesitará.

La aproximación por muestreo se puede hacer de muchas maneras, respecto a la muestra más cercana, respecto a la siguiente muestra, a la anterior… en este caso se hace respecto a la siguiente muestra. Por tanto, si las muestras son para 0º, 30º, 60º, 90º, etc. los ángulos 28º, 29º, 30º serán aproximados al valor de 30º y los ángulos 31º, 32º, 33º serán aproximados al valor de 60º, y así sucesivamente.

En las *figuras 2.6, 2.7 y 2.8* se observan las gráficas del seno siendo aproximadas con distinto número de muestras:

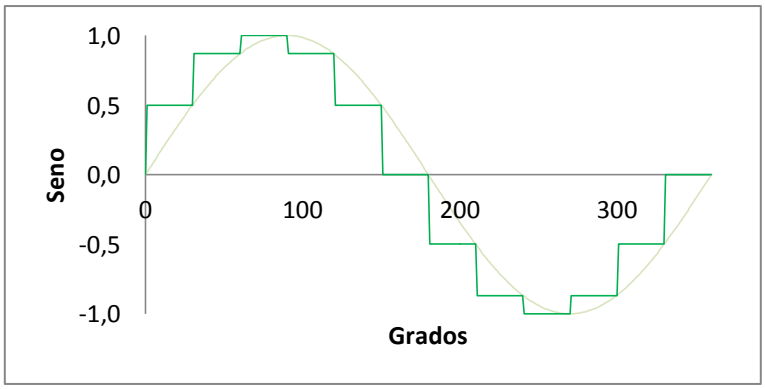


Figura 2.6. Seno aproximado mediante 13 muestras

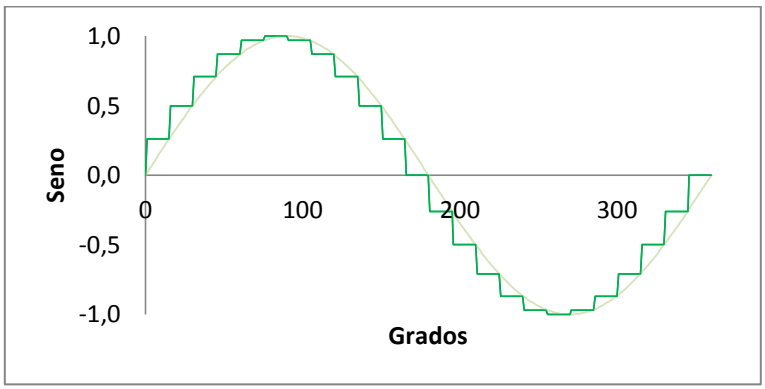


Figura 2.7. Seno aproximado mediante 25 muestras

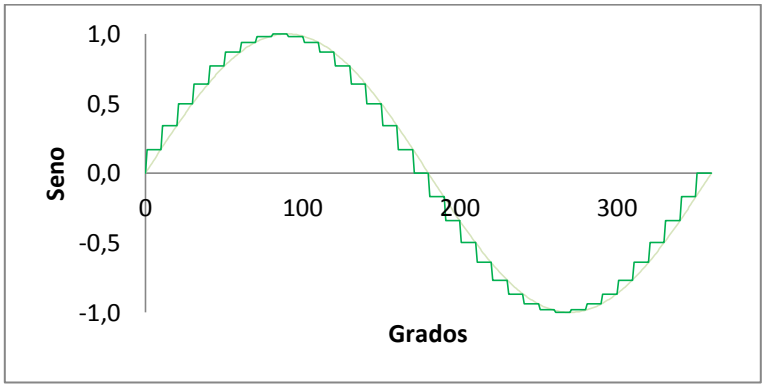


Figura 2.8. Seno aproximado mediante 37 muestras

En las figuras anteriores la representación del seno real se indica en color más claro, mientras que el color más oscuro es para la señal aproximada. Se puede apreciar perfectamente cómo al aumentar el número de muestras la diferencia entre la señal verdadera y la aproximada es menor. Por tanto, conlleva a que el error sea también más pequeño.

2.3.2. Interpolación lineal

La interpolación consiste en hallar un dato dentro de un intervalo en el que se conocen los valores en los extremos. La aproximación por muestras introduce un error demasiado elevado para operaciones de cálculo y control digital, por lo que la primera implementación que se ha considerado es la de interpolación lineal entre dos muestras. Para la aplicación la interpolación lineal calcula valores intermedios entre las muestras conocidas de la función seno, considerando que están en una línea recta, y eso supone un error (menor que para la aproximación anterior) no despreciable.

En primer lugar se explican los pasos a seguir en la interpolación. Para describir la interpolación lineal se procede a detallar un ejemplo. Partiendo de la siguiente función:

$$f(x) = (3x)^2$$

La interpolación se realiza a partir de un cierto número de puntos conocidos, por ejemplo, se tiene el valor para $x = 0, 2, 4, 6, 8, etc.$ es decir, los números pares.

Se representa la función real y se escoge un punto cualquiera como ejemplo *(x=3)*. Se procede a calcular el valor de *f(3)* a partir de los valores extremos conocidos para la interpolación, que serán dos y cuatro.

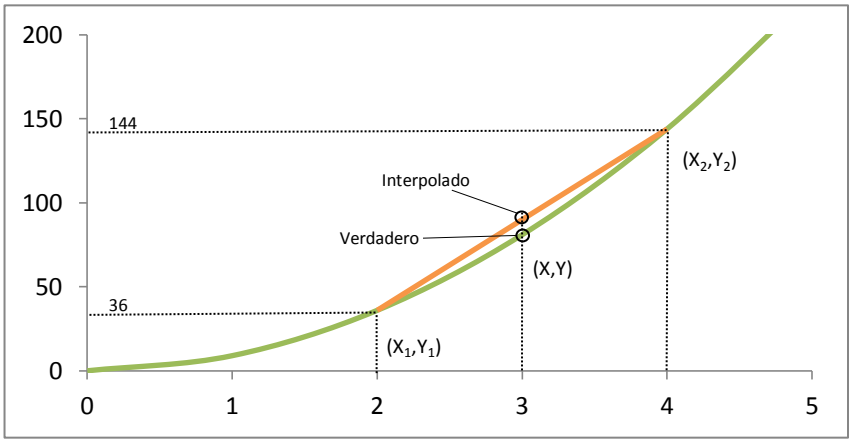


Figura 2.9. Ejemplo de interpolación

Como ya se ha explicado, una interpolación es calcular una recta entre dos puntos para así aproximar linealmente los valores que están en medio. Por tanto, se halla esa recta entre dos y cuatro, para así poder calcular tres:

$$y = mx + n$$

$$y = \frac{y_2 - y_1}{x_2 - x_1} \cdot x + n$$

El problema que surge al realizar la interpolación así, es el cálculo de *n*. Debido a que la recta usada va cambiando continuamente según los puntos escogidos, por tanto, el continúo cálculo de *n* se complica, y más aún a la hora de describirlo mediante hardware. Para solucionar esto, se hace la interpolación sin que influya este parámetro *n* de manera directa, solución que se consigue gracias al Teorema de Tales. Este teorema dícese así: "Si dos rectas cualesquiera se cortan por varias rectas paralelas, los segmentos determinados en una de las rectas son proporcionales a los segmentos correspondientes en la otra" (Tales de Mileto, siglo VI a. C.).

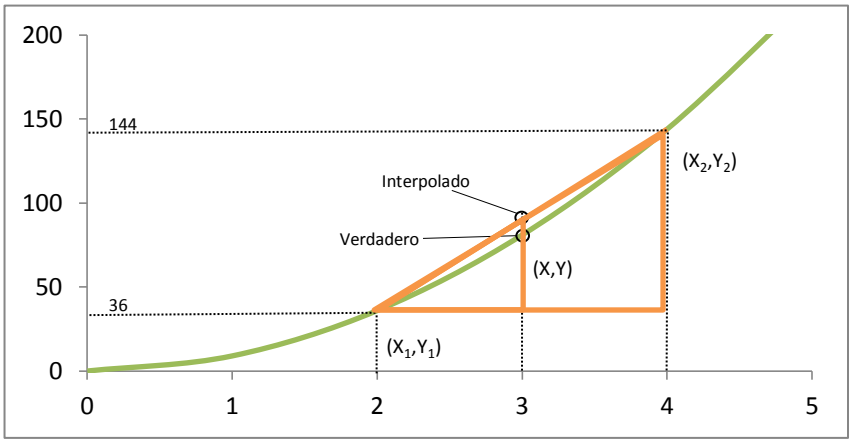


Figura 2.10. Teorema de Tales

Una vez visto esto, se puede empezar a desarrollar el teorema para extraer el resultado buscado:

$$\frac{y_2 - y}{x_2 - x} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$y_2 - y = \frac{y_2 - y_1}{x_2 - x_1}(x_2 - x)$$

$$\boldsymbol{y = y_2 - \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_2 - x)}$$

Se puede apreciar finalmente que interpolando de esta manera, los datos que intervienen de manera directa son todos conocidos.

Una vez se tiene despejada la incógnita necesaria de la ecuación, se calcula *f(3)* para el ejemplo puesto:

$$f(3) = y = y_2 - \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_2 - x) = 144 - \left(\frac{144 - 36}{4 - 2}\right)(4 - 3) = 90$$

Una vez conocido el resultado de la interpolación se compara con el resultado real que se tendria que obtener:

$$f(3) = (3x)^2 = (3 \cdot 3)^2 = 81$$

Se aprecia que el resultado no es exacto, como ya se esperaba, y como además, se podia confirmar con la gráfica antes de hacer la interpolación. Sin embargo, respecto al método de aproximación por muestreo se ve que es mucho más exacto.

Interpolación del seno

Aplicando la interpolación lineal a la función seno, cuando se tiene un número finito de muestras, el error cometido es menor que con la aproximación por muestreo. Para poder tener una equivalencia a la hora de estudiar resultados, se han dibujado las nuevas funciones seno para 13, 25 y 37 muestras. Obteniendo así, tres gráficas nuevas, *figuras 2.11, 2.12 y 2.13*. En este caso también el color claro corresponde al seno real y el oscuro al interpolado. El error se ha reducido considerablemente y prácticamente se superponen ambas gráficas, por tanto, apenas se aprecia la de color claro.

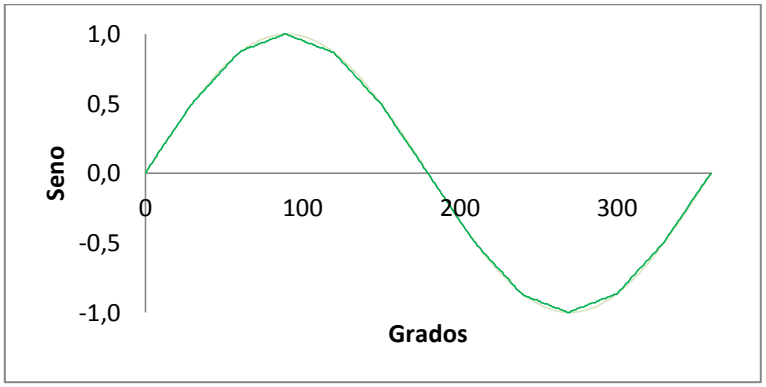


Figura 2.11. Seno interpolado mediante 13 muestras

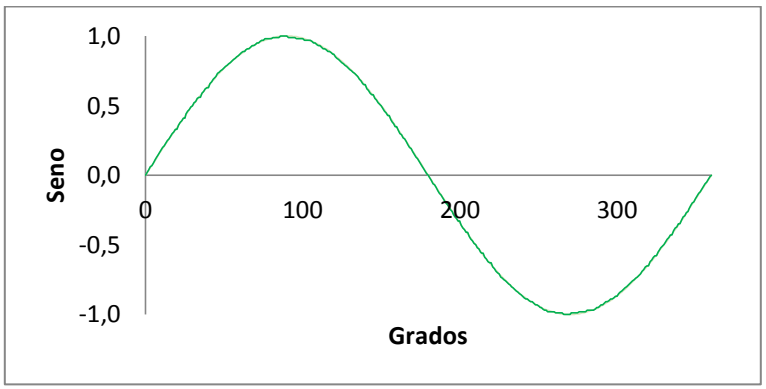


Figura 2.12. Seno interpolado mediante 25 muestras

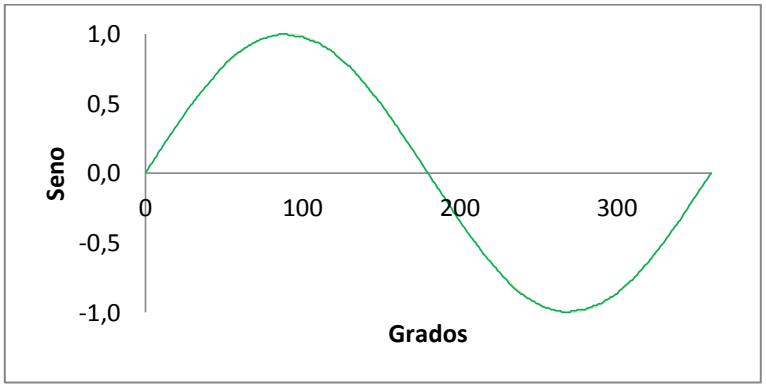


Figura 2.13. Seno interpolado mediante 37 muestras

2.3.3. Serie de Taylor

La serie de Taylor es la representación de una función mediante una suma infinita de infinitos términos. Estos se calculan a partir de las sucesivas derivadas de la función para un punto concreto. Si este punto es cero, se denomina serie de McLaurin [5].

Este método ofrece tres ventajas a la hora de usarlo, una de ellas muy importante para poder obtener un óptimo resultado en el diseño digital del proyecto:

- Las derivadas se pueden hacer término a término, por tanto, es mucho más sencillo.
- Se puede usar para calcular valores aproximados de una función.
- Si una función se puede transformar a una serie de Taylor, se puede demostrar que el resultado es el más óptimo posible.

La serie de Taylor de una función real *f(x)* infinitamente diferenciable en el entorno de un número real *a* se expresa bajo la siguiente secuencia de términos:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots$$

Aunque se puede escribir directamente mediante el siguiente sumatorio de términos:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n$$

La función que se quiere transformar en serie de Taylor es el *sen(x).* Para ello se calculan primero las sucesivas derivadas, hasta la quinta por ejemplo. Además, se hace para el valor de *x=0:*

$$\begin{array}{rcll} f(0) & = & sen(0) & = 0 \\ f'(0) & = & \cos(0) & = 1 \\ f''(0) & = & -sen(0) & = 0 \\ f'''(0) & = & -cos(0) & = -1 \\ f''''(0) & = & sen(0) & = 0 \\ f'''''(0) & = & cos(0) & = 1 \end{array}$$

Ahora una vez conocidas las derivadas, se pueden sustituir todos los valores en la serie de McLaurin:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f'''''(a)}{5!}(x-a)^5$$

$$f(x) = f(0) + \frac{f'(0)}{1!}(x-0) + \frac{f''(0)}{2!}(x-0)^2 + \cdots + \frac{f'''''(0)}{5!}(x-0)^5$$

$$f(x) = 0 + \frac{1}{1!}x + \frac{0}{2!}x^2 + \frac{(-1)}{3!}x^3 + \frac{0}{4!}x^4 + \frac{1}{5!}x^5$$

$$f(x) = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5$$

Una vez vista la sucesión que sigue la función *sen(x)* al ser transformada a serie de Taylor, se puede ampliar el número de términos a utilizar:

$$f(x) = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \frac{1}{9!}x^9 - \frac{1}{11!}x^{11} + \frac{1}{13!}x^{13} - \cdots$$

A continuación se han dibujado los resultados de distintos ejemplos, sobre todo para apreciar la influencia de usar más o menos términos. También, tener en cuenta que el valor de entrada para *f(x)* debe estar en radianes, y que además, el resultado del seno es un valor adimensional.

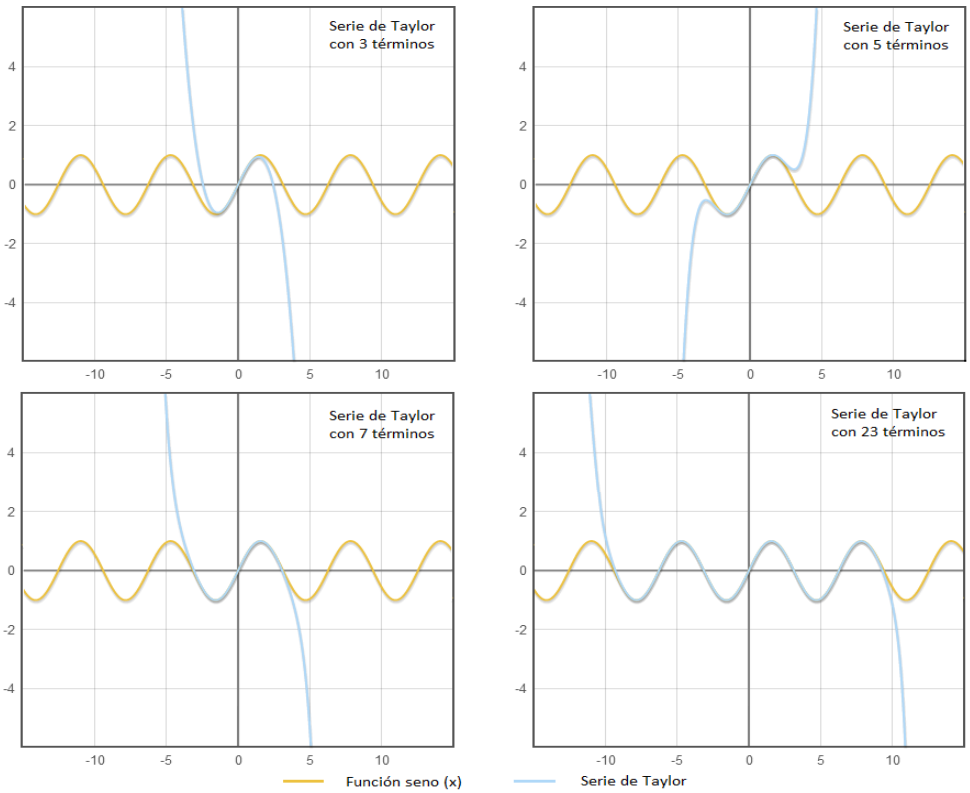


Figura 2.14. Resultados serie de Taylor para distinto nº de términos [6]

En la anterior secuencia de gráficas, se observa cómo al aumentar el número de términos, también aumenta el número de ángulos de entrada para los que la serie de Taylor ofrece una respuesta óptima, además de tener mayor precisión, aunque en la gráfica no sea visible.

En el trabajo Fin de Grado aquí expuesto se ha utilizado la serie de Taylor, para la función seno, con tres términos. Esta serie ofrece un rango de resultados óptimos en el intervalo [-90º,90º], por tanto, para calcular la función seno para el resto de ángulos posibles se estudia la relación de los cuadrantes II, III y IV con el cuadrante I. Una vez hallada esta relación, se puede obtener fácilmente el resultado del resto de ángulos (teniendo en cuenta el signo que corresponda según el cuadrante donde estemos situados).

Una vez hecho esto, los resultados obtenidos son satisfactorios y el error cometido es mínimo, como se verá más detenidamente en el capítulo 6.

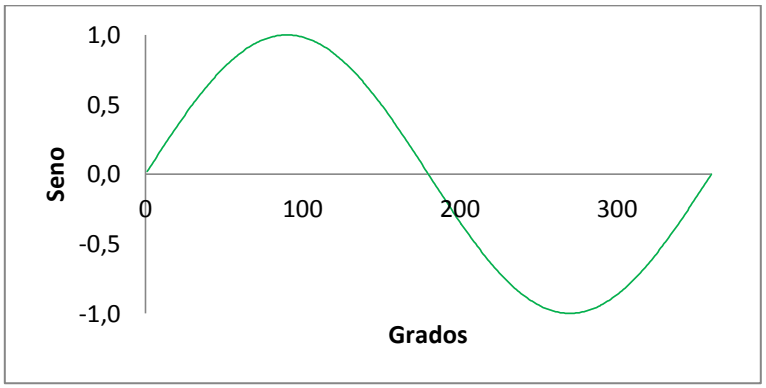


Figura 2.15. Seno mediante serie de Taylor con seis términos

Capítulo 3

Metodologías de diseño hardware

Las metodologías de diseño han facilitado el desarrollo de circuitos digitales de altas prestaciones, fiables, precisos y con el mínimo coste posible. Los dos pilares sobre los que se apoyan estas metodologías son los lenguajes de descripción de hardware, que permiten diseñar en altos niveles de abstracción, y las herramientas de síntesis lógica, que permiten obtener circuitos digitales en el nivel lógico a partir de descripciones en el nivel de transferencia de registros o superiores. Además, las herramientas de simulación de circuitos digitales en altos niveles de abstracción, con retroanotación de parámetros reales, permiten conocer de antemano el correcto funcionamiento del circuito y evitar costosos rediseños en las etapas más bajas del ciclo de diseño.

3.1. Herramientas CAD

El diseño asistido por computador, más conocido bajo las siglas inglesas CAD (*Computer-Aided Design)*, comprende el uso de diversas herramientas software para diseñar y validar, entre otros, circuitos electrónicos. Bajo estas siglas también se incluyen herramientas utilizadas por arquitectos, ingenieros y otro tipo de profesionales.

Una de las grandes ventajas de estas herramientas, es que los diseños se hacen independientemente de la tecnología final que se use en la implementación. Debido al gran coste que supone la fabricación de circuitos integrados, hay un número elevado de simulaciones en las primeras etapas, ya que cuanto más cerca estemos del prototipado, más costosa será una modificación.

3.2. Metodología de diseño *top-down.*

La metodología actual se denomina *top-down*, porque el diseño se concibe y describe en un nivel de abstracción alto, donde ya se puede simular con resultados realistas y, además, se puede sintetizar de forma automática, produciendo descripciones del circuito en niveles de abstracción más bajos, hasta llegar a un *layout* o a un *bitstream.* También existe otra metodología, conocida como *button-up*, pero que ha quedado obsoleta y no se usa desde los años 90. La *metodología top-down* contiene las siguientes fases:

- **Especificación**

En este primer apartado se enuncian los requerimientos del circuito a diseñar, además de una descripción funcional de lo que va a realizar.

- **Diseño**

Diseño arquitectural: consiste en obtener una descripción sintetizable del circuito que cumpla las especificaciones. Se realizan bancos de pruebas para corroborar el correcto funcionamiento mediante la simulación. Para ello, se divide el circuito en bloques, diseñando y validando cada uno por separado.

Diseño detallado: consiste en sintetizar las descripciones del diseño arquitectural, utilizando una herramienta de síntesis automática, capaz de generar una lista de puertas *(netlist)* que proporciona el fabricante. También se puede modificar el diseño para mejorar los resultados de síntesis, ya sea en área o tiempo.

Diseño físico: debido a que la implementación se realiza en una FPGA, este diseño consiste en generar un fichero de programación para la FPGA a partir de la lista de puertas generada en la fase anterior.

- **Programación y pruebas**

Finalmente para probar el diseño se dispone de una FPGA y de los periféricos necesarios, cuya conexión a la placa sea posible.

3.3. Descripción de un diseño

Una vez exista la idea de lo que se quiere diseñar, habrá que realizar el diseño en sí. Hay herramientas que solo permiten hacer un diseño esquemático, en los inicios las herramientas CAD eran todas así. Sin embargo, las herramientas más modernas permiten estudiar y diseñar un sistema a partir de un lenguaje funcional. Las herramientas que existen en la actualidad nos dejan realizar distintos tipos de descripciones, las cuales se explican a continuación:

- **Descripción Comportamental**

En este tipo de descripción lo importante son las entradas y las salidas. Su arquitectura interna es irrelevante, es decir, no se especifican las señales y variables de bajo nivel. Consiguiendo así, dar importancia a la salida frente a la entrada y quitándoselo al circuito físico. Por tanto, se obtienen altos resultados de diseño a muy bajo coste.

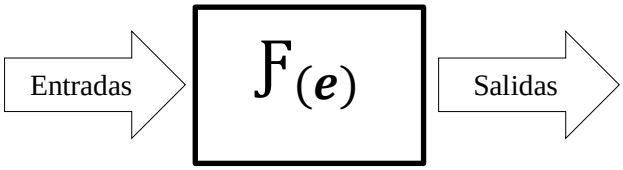


Figura 3.1. Esquema descripción comportamental

- **Descripción Estructural**

En este segundo tipo de descripción es donde toma mayor relevancia la estructura interna del diseño. Donde cada uno de los bloques internos debe poseer su propia descripción previa, todos juntos forman una organización jerárquica.

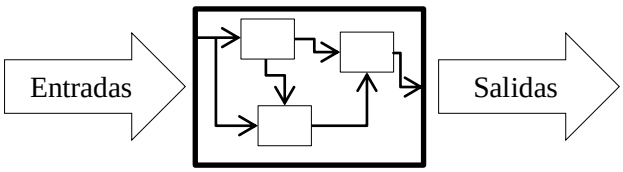


Figura 3.2. Esquema descripción estructural

- **Descripción RTL *(Resistor Transfer Logic)***

También se conoce como modelo de flujo de datos. En este último tipo de descripción se declara la sucesión temporal con la que van evolucionando las señales internas.

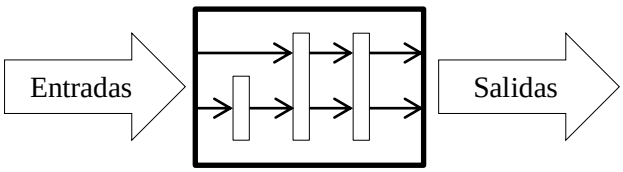


Figura 3.3. Esquema descripción RTL

Capítulo 4

Lenguaje de descripción de hardware VHDL

4.1. Introducción

En este capítulo se explica brevemente el lenguaje de descripción de hardware que se ha utilizado en este trabajo Fin de Grado, en este caso VHDL. En primer lugar se describen algunas de sus construcciones básicas, como son entidades, arquitecturas y procesos. Posteriormente, se enumeran otros elementos del lenguaje que permiten describir circuitos digitales.

El estilo de diseño de circuitos electrónicos más convencional está orientado a los componentes. El diseñador parte de un conjunto de componentes básicos con los que formará otros más complejos, y así sucesivamente, mostrando un diseño jerárquico. El lenguaje VHDL permite realizar diseños de una manera similar, y estos serán los primeros pasos de este capítulo. Sin embargo, el verdadero interés de los lenguajes de descripción de hardware es que son capaces de soportar estilos de diseño mucho más potentes, como se irá viendo más adelante.

4.2. Descripción

El VHDL nació para dar solución a numerosos problemas de diseño de circuitos de electrónicos digitales. El coste de cambiar la tecnología o las especificaciones entre un diseño y otro es más alto cuanto más cerca estemos de la fase de prototipado, más aún si esta ya hubiese sido superada. Gracias a este lenguaje eso ya no ocurre.

El VHDL ya es una ventaja en sí, pero cabe mencionar otras ventajas también realmente importantes. Una de ellas es la posibilidad de diseñar sistemas digitales independientemente de la tecnología. También cuenta con la independencia de no estar ligado a ningún tipo de simulador, sino que estos son externos [7].

4.2.1. Comportamiento

En el inicio de la descripción de un sistema digital resulta útil realizar primero una descripción en el nivel algorítmico y después, una descripción funcional en el nivel RT. Una descripción de VHDL será un conjunto de entidades ordenadas jerárquicamente. Algunas de estas pueden ser obtenidas como tal, y no ser diseñadas por uno mismo [8]. Por lo cual, es importante que tengan una descripción funcional del componente. Esta descripción solo hará referencia a lo que hace la entidad, y no a como lo hace. Muchas veces esta descripción se divide en dos, según el nivel de abstracción y del modo en que se ejecuten las instrucciones. Estas dos formas se denominan algorítmica y de flujo de datos.

4.2.2. Estructura

Un circuito electrónico digital será descrito como un módulo que consiste en una entidad y una arquitectura. En la entidad se definen los puertos de entrada y de salida, y en la arquitectura se define como se obtienen los valores de salida en función de las entradas, *figura 4.1*. Para una misma entidad puede haber varias arquitecturas que definen el comportamiento o la estructura del módulo de diferente manera.

Figura 4.1. Estructura de una entidad

A su vez cada módulo puede estar formado por submódulos, que deben instanciarse como componentes en la arquitectura de la entidad de jerarquía superior. En el esquema de la *figura 4.2* se puede ver claramente una arquitectura de este tipo.

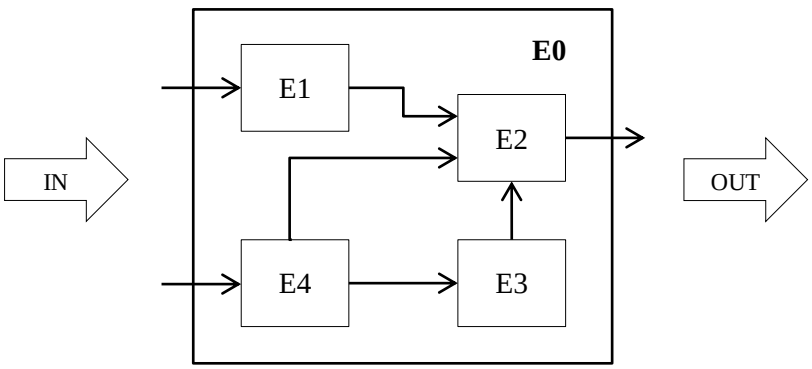


Figura 4.2. Estructura de una entidad con componentes

4.3. Conceptos básicos

Antes de comenzar con los elementos básicos de un diseño mediante lenguaje hardware VHDL conviene señalar algunos aspectos típicos de la sintaxis de este lenguaje:

- Los compiladores no distinguen entre mayúsculas y minúsculas, lo que si se recomienda, es seguir una estructura en el código para facilitar su lectura.
- Las palabras clave no se pueden usar para denominar elementos definidos por el usuario, por ejemplo, la palabra NOT no se puede usar para nombrar un puerto. Sin embargo, si se puede nombrar igual dos elementos distintos, siempre que se diferencien por el contexto, por ejemplo, una entidad que se llama SUM y un puerto de la entidad también.
- Los comentarios se hacen con dos guiones y ocupan toda la línea.
- Las líneas de texto se pueden partir según se quiera, ya que el separador verdadero que entiende el compilador es el punto y coma. Por tanto, cada sentencia acaba cuando el compilador se encuentre un punto y coma.

4.3.1. Entidades y arquitecturas

A la hora de diseñar un circuito hay dos aspectos principales y necesarios, uno es la interfaz externa y otro la funcionalidad del circuito. El primero consta principalmente de los pines de entrada y salida (entidad), mientras que el segundo se refiere a lo que hace el circuito en sí (arquitectura). A continuación vemos cómo se declaran ambos [9].

```
-- Ejemplo declaración de una entidad
ENTITY mul IS
       PORT   (a:    IN BIT;
               b:    IN BIT;
               s:    OUT BIT);
END mul;
```

Figura 4.3. Ejemplo declaración de una entidad

Tal y como se aprecia en la figura anterior, para declarar una entidad se pone primero la palabra clave ENTITY seguido del nombre y de la palabra clave IS. Luego los puertos de entrada y salida se definen entre paréntesis *(nombre: tipo_puerto tipo_dato)* y mediante la palabra clave PORT. Los puertos podrán ser de entrada (IN), salida (OUT) y entrada/salida (INOUT), seguidos del tipo de dato. Finalmente se cierra la entidad con la palabra END seguido del nombre de esta.

```
-- Ejemplo declaración de una arquitectura
ARCHITECTURE function OF mul IS
       -- Declaraciones
BEGIN
       -- Operaciones
       s <= a AND b;
END function;
```

Figura 4.4. Ejemplo declaración de una arquitectura

Una arquitectura se construye con la palabra clave ARCHITECTURE seguido del nombre, de la palabra clave OF con el nombre de la entidad a la que pertenece y luego IS. Primero se hacen las declaraciones necesarias, como variables o señales intermedias que se necesiten expresamente en dicha arquitectura. Luego tras la palabra BEGIN vienen las operaciones que van a realizarse, en este caso se ha realizado una entidad bastante fácil cuyo funcionamiento es el de una AND, por tanto, se hacen las operaciones necesarias para sacar una salida en función de las entradas. Finalmente se cierra la arquitectura con la palabra clave END seguido del nombre de la arquitectura.

4.3.2. Diseño estructural

Las metodologías de diseño convencionales se basan en construir a partir de unos componentes más sencillos otros más complejos. Se suele partir de unas bibliotecas y se van diseñando sistemas más difíciles hasta llegar a nuestro sistema final. Estos sistemas más complejos suelen estar diseñados a base de componentes que están interconectados entre ellos, siguiendo siempre una jerarquía. A continuación se aprecia un ejemplo, se va a construir un sumador de dos bits a partir de dos sumadores de un bit. Por tanto, se crea primero una entidad para un sumador de un bit, y luego se instancian dos componentes de esta entidad para crear el sumador de dos bits.

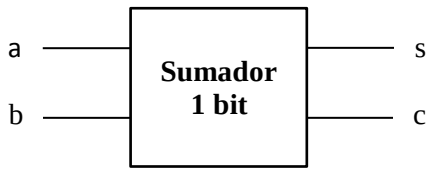


Figura 4.5. Esquema sumador de 1 bit

```
-- Declaración sumador de 1 bit
ENTITY sum_1bit IS
       PORT   (a:    IN BIT;       -- valor 1
               b:    IN BIT;       -- valor 2
               c:    OUT BIT;      -- acarreo
               s:    OUT BIT);     -- valor de la suma
END sum_1bit;

ARCHITECTURE function OF sum_1bit IS
BEGIN
       s <= a XOR b;
       c <= a AND b;
END function;
```

Figura 4.6. Código sumador de 1 bit

Una vez se tenga diseñado como funciona el sumador de un bit, lo que se hará será instanciarlo como componente en otra entidad y así crear el sumador de dos bits.

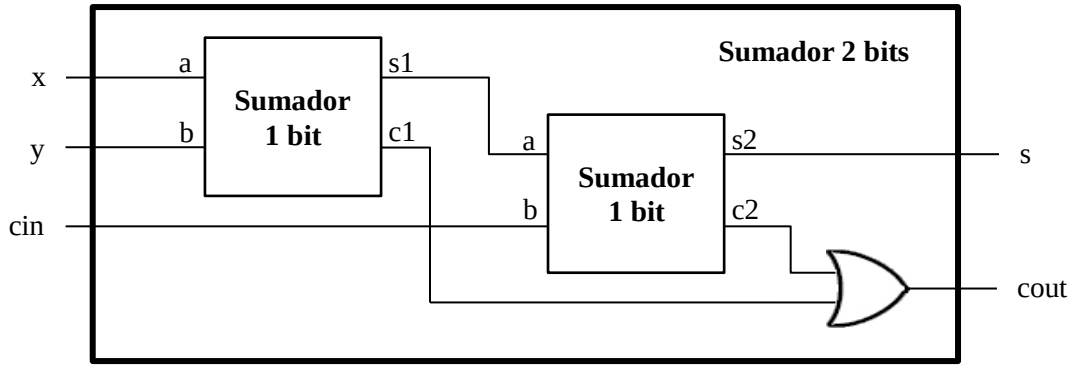


Figura 4.7. Esquema sumador de 2 bits

```
-- Declaración sumador de 2 bits
ENTITY sum_2bit IS
       PORT   (a   :       IN BIT;
               b   :       IN BIT;
               cin :       IN BIT;
               cout:       OUT BIT;
               s   :       OUT BIT);
END sum_2bit;

ARCHITECTURE function OF sum_2bit IS
       COMPONENT sum_1bit         --Declaración del componente
              PORT(  a:     IN BIT;
                     b:     IN BIT;
                     c:     OUT BIT;
                     s:     OUT BIT);
       END COMPONENT;
       SIGNAL s1, s2: BIT;
       SIGNAL c1, c2: BIT;
BEGIN
       S1:sum_1bit PORT MAP(x,y,s1,c1);        --Instanciación de
       S2:sum_1bit PORT MAP(s1,cin,s2,c2);     --los componentes
       s <= s2;
       cout <= c1 OR c2;
END function;
```

Figura 4.8. Código sumador de 2 bits

Una vez visto el ejemplo anterior es importante remarcar como se ha hecho la instanciación de los componentes. La declaración se ubica donde se declaran los elementos de la arquitectura y la instanciación después del BEGIN. Primero se le da un nombre al componente, luego dos puntos y se referencia el componente que es, se añaden las palabras clave PORT MAP y entre paréntesis y por orden respectivo, se le pasan las variables, señales o puertos que le deben entrar al componente.

4.3.3. Objetos

En VHDL hay tres tipos de objetos: constantes, variables y señales. Los dos primeros son más conocidos de los lenguajes software, mientras que las señales son un nuevo objeto que se verá detenidamente.

- **Constantes**

Tal y como su nombre indica una constante es una variable cuyo valor no va a cambiar durante la ejecución. En el siguiente ejemplo se muestra como se debe declarar este tipo de objeto:

```
CONSTANT a: BIT := '0';
```

La declaración se hace mediante la palabra clave CONSTANT seguido del nombre, dos puntos y el tipo de dato. Luego la asignación del valor se hace mediante el símbolo := y finalmente punto y coma.

- **Variables**

Una variable como tal es un objeto que podrá ir cambiando su valor a lo largo de la ejecución. En este caso, también se puede inicializar la variable al declararla, aunque esto no es sintetizable y solo serviría para la simulación. Veamos un ejemplo:

```
VARIABLE a: BIT;
```

Se declara con la palabra clave VARIABLE seguido del nombre, dos puntos y el tipo de dato. Si se quisiera inicializar a algún valor solo habría que hacerlo bajo el símbolo :=. Para cambiar el valor de una variable ha de usarse este símbolo también.

La característica principal de las variables es que solo se pueden usar en ámbitos secuenciales (procesos, funciones y procedimientos), los cuales se explican en próximos apartados. Por tanto, esas variables serán solo visibles en el proceso que hayan sido declaradas y nunca desde ámbitos concurrentes, estos también se explican más adelante.

- **Señales**

Las señales son objetos que no existen en el lenguaje software, ya que han sido creadas explícitamente para almacenar valores propios del hardware. Al igual que las variables, las señales también pueden cambiar su valor, y además, son visibles en cualquier ámbito, tanto concurrente como secuencial. También se puede inicializar a un valor, pero tampoco es sintetizable y también se hace con el símbolo :=. Como es lógico, en el hardware real las señales no están inicializadas a nada. Lo que si se hace, es usar una señal externa *(reset)* para inicializar los elementos deseados. Un ejemplo de declaración de señal sería:

```
SIGNAL a: BIT;
```

Se declara con la palabra clave SIGNAL seguido del nombre, dos puntos y el tipo de dato. Si posteriormente se quiere cambiar el valor de la señal se hará mediante el símbolo <=, en este caso se usa un símbolo distinto al usado para inicializar.

Las señales se declaran en ámbitos concurrentes, sin embargo, son visibles en cualquier ámbito. Gracias a esto forman el mecanismo principal de comunicación entre sentencias concurrentes y secuenciales.

```
ARCHITECTURE a OF b IS
       SIGNAL s1,s2,s3: BIT;
BEGIN
       s1 <= '0';
       PROCESS(s1,s2)
              VARIABLE a: BIT;
       BEGIN
              a  := s1;
              s2 <= not a;
       END PROCESS;
       s3 <= s2;
END a;
```

Figura 4.9. Ejemplo del uso de señales

Tal y como se aprecia en el ejemplo, las señales pueden modificar su valor tanto dentro como fuera del proceso, es decir, en cualquier ámbito. Sin embargo, se deben considerar una serie de aspectos a la hora de asignar valores a señales.

- Si se asigna un valor a una señal en un ámbito concurrente, esta se actualiza de inmediato.
- Si se asigna un valor a una señal en un ámbito secuencial, esta se actualiza una vez acabe ese ámbito secuencial, normalmente al acabar el proceso.
- Sin embargo, si se asigna un valor a una variable en el ámbito secuencial, esta si se actualiza de inmediato.

Con el ejemplo de la *figura 4.10* se ven las consideraciones anteriores como funcionarían:

```
ARCHITECTURE a OF b IS
       SIGNAL s1,s2,s3: BIT;
BEGIN
       s1 <= '0';          -- s1 = '0' (actualiza inmediato)
       s2 <= '0';          -- s2 = '0' (actualiza inmediato)
       PROCESS(s1,s2)
              VARIABLE a: BIT;
       BEGIN
              a  := s1;    -- a  = '0' (actualiza inmediato)
              s2 <= not a; -- s2 = '0' (no actualiza todavia)
       END PROCESS;        -- s2 = '1' (actualiza ahora)
       s3 <= s2;           -- s3 = '1' (actualiza inmediato)
END a;
```

Figura 4.10. Consideraciones en el uso de señales

4.3.4. Sentencias concurrentes y secuenciales

El lenguaje software presenta su código con una estructura secuencial, es decir, las líneas se ejecutan una detrás de otra. Sin embargo, la descripción de lenguaje de hardware conlleva una nueva dificultad. El hardware se ejecuta totalmente en paralelo, por tanto, se necesita que el programa trabaje con todos sus datos al mismo tiempo. Esto se debe a que un circuito electrónico digital tiene puertos y componentes que funcionan concurrentemente en el tiempo, es decir, a la vez. Por tanto, el VHDL debe permitir el modelado de elementos concurrentes para soportar la descripción precisa y de manera correcta. Un ejemplo muy simple:

```
ARCHITECTURE a OF b IS
       SIGNAL a,b,s: BIT;
BEGIN
       s <= a XOR b;
       a <= NOT b;
END a;
```

Figura 4.11. Sentencias concurrentes

Como muestra el ejemplo, da igual que *a* cambie después de *s*, ya que ambas se están ejecutando a la vez, y por tanto, cambiarán a la vez.

Por otro lado se tiene, que también se pueden describir procesos mediante sentencias secuenciales, donde aquí sí se sigue un orden de ejecución, que simplemente es una línea tras otra. Las variables son locales a esos procesos y las señales solo se actualizan al finalizar estos. Estas sentencias secuenciales solo se pueden hacer en procesos, funciones y procedimientos.

```
ARCHITECTURE a OF b IS
       SIGNAL a,b,s: BIT;
BEGIN
       PROCESS(a,b)
              --Declaraciones locales al proceso
       BEGIN
              a <= NOT b;
              s <= a XOR b;
       END PROCESS;
END a;
```

Figura 4.12. Sentencias secuenciales

En la *figura 4.12* se ve que el resultado de *s* sería el mismo que en *la figura 4.11*, sin embargo, las sentencias a seguir son totalmente diferentes. Si se hace de la segunda manera, el ser humano lo entiende de forma más fácil, ya que está hecho para hacer las cosas una tras otra, puesto que el cerebro humano no es capaz de pensar dos cosas a la vez.

A continuación se explica más detalladamente las distintas formas en las que podemos crear sentencias secuenciales:

- **Procesos**

Un proceso se define tal y como se ha visto en el anterior ejemplo, lo único que queda por aclarar es el tema de la lista de sensibilidad. Esta lista es lo que aparece entre paréntesis justo después de la palabra PROCESS. Con esto lo que se consigue es que el proceso se ejecuta solo cuando uno de los valores de esa lista cambie, por tanto, el proceso no se está ejecutando todo el rato, solo cuando es necesario.

- **Funciones**

Una función es un subprograma al cual le entran unos parámetros de entrada y saca otro de salida mediante la sentencia RETURN. Los argumentos de una función siempre son de entrada (IN). Una función no tiene efectos colaterales, es decir, no puede hacer cambios en objetos externos a ella. Las funciones se pueden declarar en la arquitectura, aunque normalmente se utilizan los denominados *packages* para ello.

En el siguiente ejemplo hay una simple suma de un bit mediante una función. Al solo poder devolver un valor, devuelve la suma sin tener en cuenta el posible acarreo.

```
ENTITY sum IS
       PORT   (x,y: IN BIT;
               S  : OUT BIT);
END sum;

ARCHITECTURE a OF sum IS
       FUNCTION sumar (a,b: BIT) RETURN BIT IS
              VARIABLE resultado: BIT;
       BEGIN
              resultado := a OR b;
              RETURN resultado;
       END FUNCTION;
BEGIN
       s <= sumar(x,y);
END a;
```

Figura 4.13. Ejemplo de función sumar

- **Procedimientos**

Un procedimiento es también un subprograma que nos devuelve unos valores a partir de unos argumentos de entrada. Sin embargo, un procedimiento solo puede devolver valores a través de los parámetros que se le pasen, aunque es cierto, que puede ser más de uno, no como sucede con una función. Ahora los valores de entrada podrán ser IN, OUT e INOUT. Se puede hacer cambios en valores externos al procedimiento y además, no hace falta el uso de la sentencia RETURN.

Con un ejemplo similar al anterior se aprecia, lo único que ahora si se tiene en cuenta el acarreo, ya que un procedimiento puede retornar más de un parámetro.

```
ENTITY sum IS
       PORT   (x,y,c : IN BIT;
               s,cout: OUT BIT);
END sum;

ARCHITECTURE a OF sum IS
       PROCEDURE sumar ( x1,y1,c1: IN BIT;
                         s1,cout1: OUT BIT) IS
       BEGIN
              s1 := x1 OR y1;
              cout1 := (x1 AND y1)OR(y1 AND c1)OR(c1 AND x1);
       END PROCEDURE;
BEGIN
       p: process(x,y,c)
              VARIABLE j,k,l,m,n: BIT;
       BEGIN
              j:=x; k:=y; l:=c; m:=s; n:=cout;
              sumar(j,k,l,m,n);
              s <= m;       --Los valores que se han modificado
              cout <= n;    --dentro del procedimiento se conservan
       END PROCESS;         --una vez estoy fuera de este
END a;
```

Figura 4.14. Ejemplo del procedimiento sumar

4.3.5. Diseño genérico

Cuando se crea una entidad hay que darle una longitud específica a los vectores de los puertos. Sin embargo, esta longitud no tiene por qué ser fija, sino que se puede variar según convenga. Esto resulta bastante útil a la hora de instanciar componentes, por ejemplo, un contador [10]. Si un diseño necesita usar varios contadores y cada uno de un tamaño, no hace falta realizar más de un contador, simplemente basta con realizar un solo contador y hacer genérico su tamaño. Entonces, lo único que hay que hacer a la hora de instanciarlo como componente de otra entidad es especificar el tamaño deseado. La descripción en VHDL sería la siguiente:

```
ENTITY count IS
       GENERIC(N : integer := 8 );
       PORT(  clk    : in std_logic;
              reset  : in std_logic;
              cuenta : out std_logic_vector (N-1 DOWNTO 0));
END COUNT;

ARCHITECTURE count OF count IS
BEGIN
       PROCESS(Clk, Reset)
              VARIABLE c : STD_LOGIC_VECTOR(N-1 DOWNTO 0));
       BEGIN
              IF reset = '1' THEN
                     FOR i IN 0 TO N-1 LOOP
                            c(i) <= '0';
                     END LOOP;
              ELSIF Clk'EVENT AND clk = '1' THEN
                     c <= c + '1';
              END IF;
              cuenta <= c;
       END PROCESS;
END COUNT;
```

Figura 4.15. Ejemplo de diseño genérico en un contador de N-bits

Aunque el ejemplo puesto contiene muchas cosas todavía no vistas y que se verán más adelante, es un ejemplo bastante acorde para explicar qué es el diseño genérico. Si no se especificara un valor a *N* al instanciar el contador, esta cogería 8 por defecto, que ha sido el predeterminado. La instanciación se ha visto anteriormente, y con un parámetro genérico es similar.

```
ARCHITECTURE count10 OF count10 IS
       --Declaración del componente genérico count
       SIGNAL x : STD_LOGIC_VECTOR (9 DOWNTO 0);
BEGIN
       …
       c10 : count GENERIC MAP (10) PORT MAP(clk,reset,x);
       …
END count10;
```

Figura 4.16. Instanciación de componente con parámetro genérico

4.3.6. Paquetes

En los lenguajes de descripción software aparecen agrupaciones de datos en ficheros, librerías, etc. En VHDL esto se consigue mediante el uso de paquetes. Hay paquetes donde se almacenan constantes, variables, funciones… y que luego se usan allí donde sea necesario. Esta funcionalidad se consigue a través de una construcción específica formada por la palabra clave PACKAGE, seguida del nombre del paquete y de la palabra clave IS. A continuación se especifican las declaraciones necesarias y luego se acaba con la palabra END. Después viene el cuerpo del paquete cuya construcción se forma mediante la palabra clave PACKAGE BODY, seguida del nombre del paquete y de la palabra clave IS. Aquí se hacen normalmente la definición de las funciones, dar valores a las constantes, etc. Finaliza con la sentencia END. En resumen, los paquetes tendrán dos partes, la primera para declarar y la segunda para describir. En el siguiente esquema se aprecian ambas:

```
PACKAGE iconos IS
      --Declaración de constantes, variables…
      --Declaracion de funciones, procedimientos…
END iconos;

PACKAGE iconos IS
      --Definición de constantes, variables…
      --Descripción de funciones, procedimientos…
END iconos;
```

Figura 4.17. Paquete

Para usar este paquete en la entidad requerida, hay que añadirlo al principio del todo, junto con el resto de bibliotecas. Esto se consigue mediante la palabra clave USE, por ejemplo:

```
USE WORK.iconos.ALL;
USE WORK.iconos.icono_marciano;
```

La primera sentencia indica que se ha añadido todo lo contenido en el paquete. Mientras que la segunda indica que solo se ha añadido del paquete el icono_marciano.

4.3.7. Bibliotecas

Las bibliotecas tienen la misma funcionalidad que podrían tener dentro de un entorno de desarrollo software. Es un lugar donde se guarda todo tipo de información que esté relacionada con un diseño determinado. Un ejemplo de biblioteca general es la *Library IEEE*, donde los paquetes más usados que contiene son:

- IEEE.std_logic_1164.all → Vectores
- IEEE.std_logic_arith.all → Signo, sin signo, operaciones
- IEEE.std_logic_signed.all → Vector con signo
- IEEE.std_logic_unsigned.all → Vector sin signo
- IEEE.std_logic_textio.all → Ficheros

4.4. Tipos de datos y operadores

Hasta ahora se han resueltos los ejemplos vistos con datos del tipo BIT. Sin embargo, VHDL soporta un gran número de tipos de datos los cuales sirven para desarrollar sistemas más complejos, los más relevantes son los siguientes.

4.4.1. Enteros

Este dato se denomina INTEGER en VHDL. Los objetos de tipo entero se pueden declarar de la siguiente manera:

```
CONSTANT meses: INTEGER := 12;
SIGNAL k,l: INTEGER;
```

Este tipo de dato no es sintetizable como tal, sin embargo, lo que se obtiene es un vector de bits que representa dicho entero. El valor máximo que se puede obtener es de 32 o 64 bits dependiendo del ancho de palabra del microprocesador del ordenador utilizado.

4.4.2. Reales

Estos datos sirven para representar números en punto flotante y se definen con la palabra clave REAL en VHDL. Por ejemplo:

```
CONSTANT pi: REAL := 3.141592;
```

Este tipo tampoco es sintetizable por casi ninguna herramienta, esto se debe a que trabajar con coma flotante es bastante más complicado, muy versátil y requiere mucho más hardware, por tanto, implica mucho más dinero.

4.4.3. Físicos

Un tipo de dato físico se puede resumir como un dato de tipo entero pero con unidades. Esto se debe a la necesidad de interactuar con el mundo analógico. Uno de los datos físicos más interesantes es el predefinido TIME. Ya que se usa para medir el tiempo en las simulaciones. Se declara así:

```
CONSTANT periodo_clock: TIME := 10 NS;
```

Las unidades del tipo físico TIME se muestran a continuación:

FS = 10^{-15}seg.	US = 10^{-3}seg.
PS = 10^{-12}seg.	SEC = 1 seg.
NS = 10^{-9}seg.	MIN = 60 seg.
US = 10^{-6}seg.	HR = 60 min.

Los datos de tipo físico no son sintetizables. Como es lógico, en el caso TIME, no es posible obligar a un sintetizador a que un circuito o una puerta lógica sintetice con un retardo determinado, ya que esto se resuelve con otras herramientas.

4.4.4. Enumerados

Un tipo enumerado es aquel dato que se define mediante una enumeración de valores o caracteres que lo identifican. Al sintetizar la herramienta no entiende como tal esos caracteres que lo identifican, por tanto lo que hace es dar a cada carácter un valor binario diferente, el cual lo represente. A continuación hay varios ejemplos:

```
TYPE semáforo IS(verde,ambar,rojo);
TYPE estados IS(inicio,subir,bajar);
```

Una vez se tiene el tipo declarado, se declaran objetos que sean de ese tipo, por ejemplo:

```
VARIABLE sem1:semáforo;
SIGNAL estado_actual:estados;
…
sem1 := ambar;
estado_actual <= bajar;
```

Gracias a este tipo de datos se facilita la comprensión del lenguaje, ya que es más cómodo diseñar un semáforo mediante los tres tipos de estados en que puede estar, que asignándole los códigos “00”, ”01” y “10”.

A parte de los tipos enumerados que uno mismo puede crear, en VHDL ya existen algunos tipos enumerados predefinidos. Como es el caso del tipo BOOLEAN, que puede tomar dos valores solo: ‘0’ o ‘1’. También existe el tipo BIT (STD_LOGIC), que sin embargo, es mucho más complejo y puede tomar un mayor número de valores, los cuales están estandarizados bajo la normal IEEE 1164 y se muestran en la siguiente tabla:

Valores del tipo lógico estándar STD_LOGIC		
Fuertes	‘U’	No inicializado
	‘X’	Desconocido
	‘0’	Cero lógico
	‘1’	Uno lógico
	‘Z’	Alta impedancia
Débiles	‘W’	Desconocido débil
	‘L’	Cero débil
	‘H’	Uno débil
Otros	‘-‘	Indiferentes

Tabla 4.1. Valores del tipo STD_LOGIC

Aunque STD_LOGIC es un tipo predefinido, no se encuentra en el estándar del lenguaje, sino que viene incluido en la biblioteca IEEE.std_logic_1164.ALL.

4.4.5. Vectores

Un vector es un tipo compuesto cuyos elementos son del mismo tipo. En VHDL estos elementos son del tipo STD_LOGIC. Por tanto, el vector pasa a ser del tipo STD_LOGIC_VECTOR. Dentro de VHDL los vectores cobran gran relevancia, debido a que son el arma fundamental para afrontar todos los problemas que pueda llevar a cabo el diseño de un circuito electrónico digital.

Su notación debe ir entre comillas y al ser frecuente que estos vectores sean de gran tamaño, pueden anotarse en base binaria, octal y hexadecimal. También pueden incluir "_" para separarlos en partes y obtener una lectura más fácil. Así, los siguientes ejemplos denotan el mismo valor:

```
"100010101011"      -- Binario
B"100010101011"     -- Binario
B"1000_1010_1011"   -- Binario separado con "_"
O"4253"             -- Octal
X"AB"               -- Hexadecimal
```

A la hora de declarar un vector se debe de indicar la longitud de este, se puede hacer de izquierda a derecha o de derecha a izquierda:

```
CONSTANT a: STD_LOGIC_VECTOR(7 DOWNTO 0):="00010110";
CONSTANT b: STD_LOGIC_VECTOR(0 TO 7):="01101000";
```

También puede accederse a una posición concreta o a un rango de posiciones, por ejemplo:

```
a(3);               -- La posicion a(3) vale '1'
a(5 DOWNTO 1);      -- El vector de 5 bits vale "01011"
```

Una vez conocido todo esto, ahora puede hacerse un vector de vectores, obteniendo así una matriz. La única objeción que surge, es que a una matriz solo se puede acceder a cada posición de manera individual o a ella en todo su conjunto. Por ejemplo:

```
TYPE matriz IS ARRAY(0 TO 255,0 TO 255) OF STD_LOGIC;
SIGNAL imagen: matriz;

TYPE memoria1 IS ARRAY(0 TO 20) OF STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL ram1: memoria;

TYPE memoria2 IS ARRAY(0 TO 255) OF INTEGER RANGE (0 TO 1023);
SIGNAL ram2: memoria;
```

Si se quiere acceder y modificar el valor de una posición concreta se haría de la siguiente manera:

```
imagen(239)(54) <= '1';
ram1(15) <= "0100";
ram2(25) <= 524;
```

4.4.6. Operadores

Dentro del lenguaje hardware hay un amplio número de operadores con los que se puede trabajar, en la siguiente tabla se puede ver un resumen de ellos.

Operadores	Descripción	Notación	Operandos	Resultado
Lógicos	NOT OR NOR AND NAND XOR XNOR	NOT OR NOR AND NAND XOR XNOR	BIT BOOLEAN STD_LOGIC_VECTOR STD_LOGIC STD_LOGIC_VECTOR BIT_VECTOR STD_LOGIC_VECTOR	Ídem
Relacionales	Igual Distinto Mayor que Menor que Mayor o igual Menor o igual	= /= > < >= <=	Cualquiera	BOOLEAN
Aritméticos	Suma Resta Multiplicación División Resto división Módulo exponenciación	+ - * / REM MOD **	INTEGER REAL FÍSICO STD_LOGIC_VECTOR STD_LOGIC_VECTOR STD_LOGIC_VECTOR STD_LOGIC_VECTOR	Ídem
Desplazamiento y rotación	desp. log. izda. desp. log. dcha. desp. arit. izda. desp. arit. dcha. rotación izda. rotación dcha.	SLL SRL SLA SRA ROL ROR	STD_LOGIC_VECTOR	Ídem
Concatenación	Concatenación	&	Vectores y bits	Vector

Tabla 4.2. Principales operadores

4.4.7. Atributos

Los atributos denotan valores, funciones, tipo o rangos asociados a distintos elementos del lenguaje. Existe un gran número, sin embargo, se verán los principales. Los atributos predefinidos pueden ser de tipo, señal o vector.

El atributo de 'tipo' permite obtener información acerca de determinadas propiedades de ese tipo. No son sintetizables.

El atributo de 'señal' que se va a ver es solo uno: 'EVENT. El cual es de tipo BOOLEAN, y devuelve TRUE si la señal a la que se aplica esta activa en ese momento y FALSE en caso contrario. Este atributo es de vital importancia a la hora de poder usar

los biestables, ya que con él se manejan los flancos de reloj. Estos flancos pueden ser de subida o de bajada, y se conrresponden con la siguiente sentencia:

```
CLK´EVENT AND CLK = '1'  -- Flanco de subida
CLK´EVENT AND CLK = '0'  -- Flanco de bajada
```

Finalmente hablar de los atributos de vector. Estos dan información del vector sobre el que son aplicados. Partiendo de un STD_LOGIC_VECTOR (15 DOWNTO 0), en la siguiente tabla se muestran los distintos atributos de vector:

Atributo	**Valor obtenido**
d'LEFT	15
d'RIGHT	0
d'HIGH	15
d'LOW	0
d'RANGE	15 DOWNTO 0
d'REVERSE_RANGE	0 TO 15
d'LENGTH	16

Tabla 4.3. Atributos de vector

4.5. Estructuras combinacionales

Los circuitos combinacionales son circuitos que implementan funciones lógicas. El diseño consiste en determinar la función lógica del circuito y descomponerla en funciones más sencillas que se puedan implementar con puertas lógicas o elementos básicos, para luego ensamblarlo todo.

En el capítulo 3.3 se vieron las distintas formas de diseño que existen. El estilo de diseño estructural permite diseñar un circuito combinacional cuando su descomposición es conocida. Porque sino, esta descomposición queda en manos del diseñador, y puede llegar a ser una tarea bastante dificultosa, incluso imposible. Las modernas herramientas de síntesis existentes nos permiten obtener la implementación de estos elementos básicos con sofisticados algoritmos que ejecuta el computador. Por tanto, el diseñador simplemente tendrá que estar centrado en realizar la descripción necesaria del circuito, ya que la herramienta será quien haga la descomposición necesaria en esos elementos básicos. Esta forma de diseño se denomina comportamental, orientada a la funcionalidad del circuito y no a su contenido.

4.5.1. Sentencias condicionales

Las sentencias condicionales pueden usarse en ámbito secuencial y concurrente. Lo único que para cada ámbito se deben de usar sentencias diferentes. En el ámbito secuencial se usan las sentencias IF y CASE, mientras que en el ámbito concurrente están la asignación condicional y la asignación seleccionada. Respectivamente son equivalentes, es decir, se puede modelar un mismo comportamiento utilizando cualquiera de las dos sentencias que sean equivalentes entre sí.

Sentencia IF (secuencial) → Asignación condicional (concurrente)
Sentencia CASE (secuencial) → Asignación seleccionada (concurrente)

Figura 4.18. Sentencias condicionales

4.5.2. Sentencias condicionales: Secuenciales

El funcionamiento de estas sentencias es igual al funcionamiento que realizan los multiplexores, de hecho, se sinterizarán en multiplexores de infinitos tipos.

- **Sentencia IF**

La sentencia IF debe de ir dentro de un proceso, el cual estará dentro de una arquitectura. Se construye con la palabra clave IF, seguido de la condición o condiciones que se quiera, y luego THEN. Después se añaden las líneas de código. Se pueden anidar los IF mediante la palabra clave ELSIF que sigue la misma estructura que el IF. Si no se cumple ninguna condición se puede añadir ELSE, pero este no es estrictamente necesario. Finalmente se tiene que cerrar cada IF que se abra con un END IF.

```
PROCESS(a,b,c)
BEGIN
       IF a = '1' THEN
              y <= a;
       ELSIF b = '1' THEN
              y <= b;
       ELSE
              z <= c;
       END IF;
END PROCESS;
```

Figura 4.19. Sentencia IF

- **Sentencia CASE**

Cuando hay múltiples opciones y además todas dependen de la misma variable, en vez de anidar varios IF, se usa esta sentencia. Se construye con la palabra clave CASE, seguido de la variable y luego IS. Ahora se enumeran los distintos resultados que puede tener la variable. Se pone WHEN, seguido del resultado y luego =>, ahora se añade el código que se debe ejecutar si se cumple la condición. Finalmente se cierra la sentencia con END CASE.

```
PROCESS(a,b,c,d,z)
BEGIN
       CASE z IS
              WHEN (0) => s <= a;
              WHEN (1) => s <= b;
              WHEN (2) => s <= c;
              WHEN (3) => s <= d;
              WHEN OTHERS => s <= 0;
       END CASE;
END PROCESS;
```

Figura 4.20. Sentencia CASE

4.5.3. Sentencias condicionales: Concurrentes

El funcionamiento para sentencias condicionales concurrentes se hace de forma diferente pero se obtiene el mismo resultado. Es decir, se hacen multiplexores de otra manera. Hay que tener en cuenta que ahora es un ámbito concurrente, por tanto, se opera directamente en la arquitectura y nunca dentro de un proceso.

- **Asignación condicional**

En este apartado se crea un multiplexor simple mediante una sentencia condicional concurrente.

```
ARCHITECTURE a OF mux IS
     SIGNAL a,b,c,s: STD_LOGIC;
BEGIN
     …
     s <= a WHEN c = '0' ELSE b;
     …
END a;
```

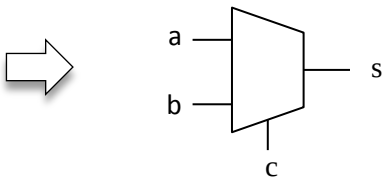


Figura 4.21. Asignación condicional

- **Asignación seleccionada**

De igual modo que hacía el CASE cuando hay múltiples posibilidades, ahora lo hace la asignación seleccionada. Se construye con la palabra clave WITH, seguido de la variable y luego SELECT. Después la asignación de valores se hace igual que en el ejemplo de asignación condicional, pero ahora con varias posibilidades.

```
WITH c SELECT
s <=  a WHEN "00",
      b WHEN "01",
      c WHEN "10",
      d WHEN "11",
      '0' WHEN OTHERS;
```

Figura 4.22. Asignación seleccionada

4.5.4. Reglas para el diseño de circuitos combinacionales

En primer lugar decir que la síntesis de sentencias condicionales produce siempre multiplexores. Entonces se podría pensar como reproducir otros elementos como decodificadores, etc. Hay que decir que todo circuito combinacional se puede construir a partir de multiplexores. Para un correcto diseño de estos circuitos se deben de tener en cuenta varios requisitos esenciales:

- Siempre que se asigne una señal en un ámbito concurrente, hay que asegurarse de que esa señal tenga un valor en cualquiera de los casos posibles que se puedan dar.

- La lista de sensibilidad de un proceso debe de contener todas aquellas señales cuyo valor se utiliza dentro del proceso y que por tanto actuarían cómo entradas en el circuito resultante de ese proceso.
- Si se usan variables dentro de un proceso, estas deben de ser escritas antes de ser leídas.

4.6. Estructuras secuenciales

4.6.1. Circuitos síncronos y asíncronos

Los circuitos secuenciales se caracterizan por tener realimentaciones en algunas de sus señales, consiguiendo así almacenar información para su posterior uso. Esta propiedad se considera memoria, y su elemento básico es el biestable, capaz de almacenar un bit de memoria. Los biestables son capaces de capturar información y almacenarla hasta que sea necesario actualizarla.

Un circuito secuencial contiene un gran número de biestables. El estado de estos marca el estado del circuito. Uno de los problemas que surgen son los retardos que puede haber en el circuito, lo que conlleva a que los distintos biestables no cambien de estado a la vez, creando así posibles errores. Por tanto, se deben tener perfectamente sincronizados todos los biestables de un circuito, algo que no resulta tan sencillo de hacer. Sin embargo, y después de abordar numerosos métodos, el más utilizado y a la vez el más sencillo es el estilo de diseño síncrono. Por contraposición, el resto de diseños se suelen denominar asíncronos. Un diseño síncrono se caracteriza por cumplir las siguientes condiciones:

- Todos los biestables se activan bajo la misma señal de reloj.
- Todos los biestables son activos por flanco de reloj.
- Los biestables pueden tener entradas asíncronas independientes (*clear* o *preset*).
- Todos los biestables están conectados a una señal común y externa de inicialización (*reset*).

Como se puede apreciar, en un circuito secuencial el elemento que gobierna todo el flujo de ejecución es el reloj. Todos los biestables cambian cuando se produce un flanco de reloj y guardan su estado hasta que se vuelva a producir el siguiente flanco de reloj. Para que todos los cambios que se producen en un biestable se propaguen por la parte combinacional del circuito sin que haya problemas de retardos, sincronización, etc. se tiene que elegir una frecuencia de reloj adecuada.

Gracias a los circuitos síncronos se pueden crear elementos secuenciales realmente útiles y que ofrecen una mayor gama de posibilidades a la hora de hacer circuitos electrónicos digitales. Podrán diseñarse registros, contadores, máquinas de estados, etc.

4.6.2. Biestables y registros

Como ya se ha dicho, el biestable es el elemento básico de un sistema secuencial. Mientras que en apartados anteriores se vio que un multiplexor se puede realizar de distintas maneras, aquí se verá que un biestable tiene una única estructura de diseño. Es cierto que se le pueden añadir entradas asíncronas o síncronas diferentes, ya sean *reset*, *preset*, *clear*, etc. sin embargo, la estructura en sí, no se verá modificada.

```
PROCESS(clk,reset)
BEGIN
       IF reset = '1' THEN
              Q <= '0';
       ELSIF clk'EVENT AND clk = '1' THEN
              IF enable = '1' THEN
                     Q <= d;
              END IF;
       END IF;
END PROCESS;
```

Figura 4.23. Biestable D con reset

Se puede ver que la lista de sensibilidad del proceso solo lleva *clk* y *reset*, esto se debe a que solo hay que meter en la lista el reloj y las señales asíncronas que pueda tener. En este caso hay un *reset*, si hubiera por ejemplo hubiera un *clear* asíncrono también debería de meterse. Por esta misma razón no se mete el *enable*, ya que al estar dentro de la condición de flanco de reloj, ya no es una entrada asíncrona, sino todo lo contrario (síncrona).

En la condición de flanco se aprecia como *clk = '1'*, esto es porque el biestable cambia con el flanco de subida. Si estuviera *clk = '0'*, entonces el biestable cambiaria con el flanco de bajada.

4.6.3. Reglas para el diseño de procesos secuenciales

Al igual que hay que cumplir unas reglas al diseñar estructuras combinacionales, hay que acatar otras con las estructuras secuenciales. En primer lugar destacar que todo se rige bajo la sentencia IF que indica la existencia del flanco de reloj. Antes de este IF se deberán hacer las inicializaciones asíncronas necesarias, si es que las hubiera, ya que no son obligatorias. Tras el IF viene todo el contenido síncrono de los biestables.

En la *figura 4.23* aparece un ejemplo de biestable D con *reset*, siguiendo unas pautas estructurales. Ahora se describen estas pautas en un modelo general, el cual se debe de cumplir siempre a la hora de realizar procesos secuenciales.

```
PROCESS(reloj + entradas asíncronas)
BEGIN
       IF (condición inicialización asíncrona) THEN
              --Inicialización asíncrona
       ELSIF (FLANCO ACTIVO DE RELOJ) THEN
              --Comportamiento síncrono
       END IF;
END PROCESS;
```

Figura 4.24. Modelo general para procesos secuenciales

4.6.4. Máquina de estados finitos

Uno de los ejemplos más importantes posibles de realizar gracias a los procesos secuenciales son las máquinas de estados finitos. Estas se caracterizan por tener además de entradas y salidas, un conjunto de estados internos. Según cambien las entradas, se irá a un estado u a otro. Gracias a estos se puede pensar en un diseño más optimizado de multitud de circuitos electrónicos digitales, como puede ser el funcionamiento de un semáforo, un ascensor, etc.

Para la realización de una máquina de estados mediante un circuito síncrono se suele seguir el modelo de Huffman, que se puede ver representado en el siguiente esquema:

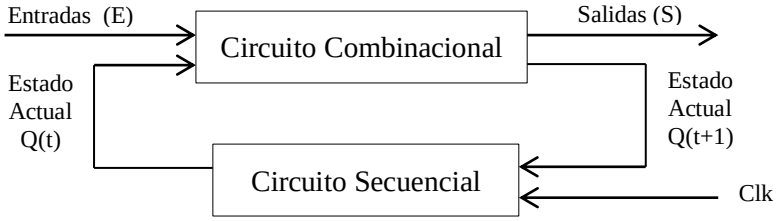


Figura 4.25. Modelo Huffman para máquina de estados finitos [9]

Una vez vistos los fundamentos de una máquina de estados, se enuncian los dos modelos existentes.

- **Máquina de Moore**

Este tipo de máquina se fundamenta en que la salida depende solo del estado en el que estemos. Donde cada línea de transición lo único que llevará es la entrada, la cual hará cambiar de estado. Tal y como se aprecia en la *figura 4.26*.

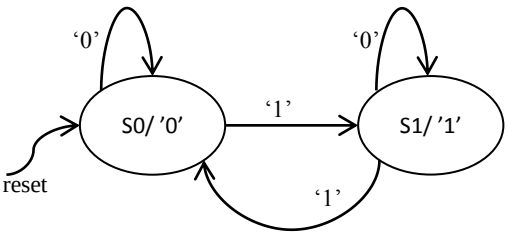


Figura 4.26. Ejemplo máquina de estados de Moore

Ahora se verá cómo sería su implementación en el lenguaje VHDL. Se evita declarar la entidad que simplemente tendría como entradas el *clock, reset* y *e*, y como salida *s*.

```
ARCHITECTURE a OF b IS
       TYPE estados IS (S0,S1);
       SIGNAL actual, siguiente: estados;
BEGIN
       PROCESS(clk,reset)
       BEGIN
              IF reset = '1' THEN
                     actual <= S0;
              ELSIF clk'EVENT AND clk = '1' THEN
                     actual <= siguiente;
              END IF;
       END PROCESS;

       PROCESS(actual,e)
       BEGIN
              CASE actual IS
              WHEN S0 =>   s <= '0';
                     IF e = '1' THEN
                            siguiente <= S1;
                     ELSE
                            siguiente <= S0;
                     END IF;
              WHEN S1 =>   s <= '1';
                     IF e = '1' THEN
                            siguiente <= S0;
                     ELSE
                            siguiente <= S1;
                     END IF;
              END CASE
       END PROCESS;
END a;
```

Figura 4.27. Máquina de Moore en VHDL

El primer proceso se utiliza para que cuando haya un flanco de reloj, se pase de un estado a otro.

El segundo proceso es donde se describe el cambio de estado. Teniendo en cuenta el estado y el valor de sus entradas, se pasa a un estado u a otro.

- **Máquina de Mealy**

Este tipo de máquina genera una salida basándose en su estado y su entrada. Por tanto, el diagrama de estados debe de incluir en cada línea de transición tanto la entrada como la salida.

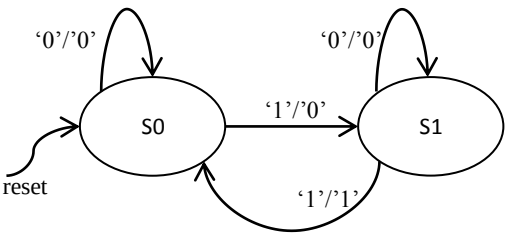


Figura 4.28. Ejemplo máquina de estados de Mealy

Al escribirlo en VHDL queda algo muy similar al anterior, de hecho el primer proceso no se repite porque es exactamente igual. Ya que este proceso como se acaba de decir, lo único que hace es pasar al estado siguiente en cada flanco de reloj. Por tanto, solo se describe el segundo proceso.

```
PROCESS(actual,e)
BEGIN
       CASE actual IS
       WHEN S0 =>
              IF e = '1' THEN
                     s <= '0';
                     siguiente <= S1;
              ELSE
                     s <= '0';
                     siguiente <= S0;
              END IF;
       WHEN S1 =>
              IF e = '1' THEN
                     s <= '1';
                     siguiente <= S0;
              ELSE
                     s <= '0';
                     siguiente <= S1;
              END IF;
       END CASE
END PROCESS;
```

Figura 4.29. Máquina de Mealy en VHDL

A pesar de la similitud, ahora se ve que la salida no cambia inmediatamente según el estado, sino que lo hace según las entradas valgan una cosa u otra.

4.7. Operaciones iterativas

En el lenguaje hardware el uso de operaciones iterativas no es tan abundante como en el lenguaje software. La explicación está en que un bucle lo que hace es repetir un número de veces lo que hay dentro. Si la descripción es software no pasa nada, porque lo que hace la herramienta es repetir código utilizando más memoria. Sin embargo, si la descripción es hardware hay un problema. Este problema se debe a que la herramienta no puede repetir hardware sin saber cuánto concretamente tiene que repetir. Por tanto, no se puede sintetizar un WHILE, ya que este no especifica el número de repeticiones. Aunque es cierto que algunos sintetizadores sí que lo aceptan, siempre que el nivel de repeticiones sea estático. Normalmente lo que se hace es sintetizar el bucle FOR, y cuando en este, esté limitado el número de iteraciones.

```
--bucle while
WHILE a < b LOOP
       …
END LOOP;

--bucle FOR
FOR i IN 0 TO 7 LOOP
       …
END LOOP;
```

Figura 4.30. Descripción en VHDL de FOR y WHILE

También existen dos sentencias que sirven para saltarse una iteración o para salir del bucle correspondiente, aunque hay que saber que tampoco son sintetizables, al igual que WHILE.

```
FOR i IN 0 TO 7 LOOP
       …
       NEXT WHEN i = 3;          --Saltamos a la siguiente iteración
       …
       EXIT WHEN i = 6;          --Salimos del bucle for
       …
END LOOP;
```

Figura 4.31. Descripción en VHDL de NEXT y EXIT

4.8. Simulación

La simulación es un tema fundamental en el desarrollo hardware, ya que como se ha ido remarcando durante todo el proyecto, ahorra mucho dinero a la hora de fabricar un circuito electrónico digital. Todo se debe a poder probar si el funcionamiento del circuito es correcto antes de prototiparlo.

4.8.1. Bancos de pruebas

En una simulación se varian las entradas para poder estudiar el resultado de las salidas. Esto se conoce como generación de estímulos. Se introduce una señal periódica al reloj del circuito que actúa de entrada y se modifican las entradas asíncronas y síncronas del circuito las veces necesarias, para ver que todos los resultados posibles son correctos.

Hay dos sentencias importantes dentro de la simulación como AFTER y WAIT, que sirven para ir cambiando las entradas según pasa el tiempo. Además, al estar hablando de tiempo tendremos que usar datos de tipo físico TIME.

El banco de pruebas es la descripción en VHDL para realizar simulaciones. Estos bancos de pruebas siguen una estructura genérica.

```
ENTITY tb IS
END tb;

ARCHITECTURE a OF tb IS
       --Declaración de los componentes a probar
       COMPONENT count
              PORT(…);
       END count;

       --Declaración de las señales
       SIGNAL a: …
       …

BEGIN
       --Instanciación del componente a probar
       uut: count PORT MAP(a,…)
       --Generación de estímulos
       …
       --Comprobación de resultados
       …
END a;
```

Figura 4.32. Estructura banco de pruebas

4.8.2. Ficheros

Igual que en otros lenguajes, cuando se desea utilizar una alta cantidad de datos y almacenarlos externamente para un uso posterior, los ficheros son la mejor solución. En los lenguajes software los ficheros se suelen aprender al principio, sin embargo, en los lenguajes hardware al no ser sintetizables puede que se hagan menos importantes. Su principal importancia viene en guardar datos obtenidos para poder interactuar con ellos en otros programas [11][12].

Para definir un fichero primero se debe declarar un nuevo tipo de datos, en este caso serán de tipo fichero.

```
TYPE file_enteros IS FILE OF INTEGER;
```

Una vez definido el tipo, se declaran los tipos de objeto fichero, que pueden ser de entrada o salida.

```
FILE file_in : file_enteros IS IN "in.dat";
FILE file_out: file_enteros IS OUT "out.dat";
```

También se definen los distintos modos de acceso que hay a la hora de abrir un fichero.

```
FILE file_in : file_enteros OPEN READ_MODE IS IN "in.dat";
FILE file_out: file_enteros OPEN WRITE_MODE IS OUT "out.dat";
FILE file_out: file_enteros OPEN APPEND_MODE IS OUT "out.dat";
```

El tipo de fichero más habitual es el fichero de texto, que contiene caracteres organizados en líneas. Los ficheros de texto tienen tipos de datos definidos, que son: STRING, LINE y TEXT. Para acceder a ellos, VHDL proporciona cinco subrutinas:

```
READLINE(fichero,línea);
WRITELINE(fichero,línea);
READ(línea,dato);
WRITE(linea,dato);
ENDFILE(fichero);
```

A continuación un ejemplo sencillo de cómo usar un fichero:

```
PROCESS
      FILE f: TEXT OPEN WRITE_MODE IS "año.txt";
      VARIABLE linea: LINE;
      VARIABLE dato: INTEGER;
      CONSTANT año: STRING := "AÑO: ";
BEGIN
      --Lectura de datos
      READLINE(input,linea);
      READ(linea,dato);
      --Escritura del dato
      WRITE(linea,año);
      WRITE(linea,dato);
      WRITELINE(f,linea);
END PROCESS;
```

Figura 4.33. Ficheros

Capítulo 5

Diseño digital de la función seno

En este capítulo se describre el diseño hardware de la función seno realizado para los distintos métodos seleccionados y explicados en el capítulo 2. En primer lugar se presentan los métodos y herramientas utilizadas durante el desarrollo del trabajo. Se describen la herramienta utilizada, el tipo de almacenamiento de datos, los imprevistos y complicaciones surgidas, las soluciones encontradas, etc.

5.1. Herramienta de desarrollo

La herramienta es un programa software CAD que permite describir el circuito digital en un alto nivel de abstracción y sintetizarlo para generar nuevas descripciones en nivel de abstracciones inferiores. Hay un gran número de fabricantes de dispositivos reconfigurables (FPGAs) que proporcionan de forma gratutira el software de diseño y prototipado para sus dispositivos. En este trabajo Fin de Grado se ha seleccionado el fabricante Xilinx™ por su amplia distribución en el mundo industrial, por la disponibilidad del software de diseño y prototipado, y por el amplio abanico de placas de desarrollo disponibles en el mercado con un coste medio-bajo.

5.1.1. Xilinx ISE Design Suite 13.2

Xilinx es una de las mayores empresas en investigación y desarrollo de FPGAs. Fue fundada por Ross Freman, el inventor de la FPGA, además de por Bernie Vonderscmitt y Jim Barnett en 1984, con su base en Silicon Valley. Hoy en día su base se encuentra en San José, California, mientras que en Europa se hallan en Dublín e Irlanda, y Singapur en Asia [13].

Xilinx desarrolla FPGAs y CPLDs que son usados en numerosas aplicaciones de telecomunicaciones, automoción, industria, etc. Las familias de dispositivos Xilins son: Virtex (alto rendimiento), Spartan (bajo coste) y Cool Runner I y II (lógica de pegamento, también conocida como el circuito lógico usado para unir circuitos integrados) [14][15]. También crea núcleos IP en lenguaje HDL, desde básicos contadores hasta complejos microcontroladores, como el Microblaze. En el pasado año, sacó al mercado su último dispositivo, la Kintex-7 FPGA, una nueva categoría de FPGAs con un rendimiento de alta gama a coste medio-bajo.

5.2. Formato de los datos

En primer lugar se habla del formato que tienen los datos de entrada y salida en la función seno a desarrollar. Nuestro objetivo es crear la función seno, y consecuentemente, tanto los datos de entrada como de salida deben ser números reales. Los datos de entrada se corresponden con el ángulo cuyo seno hay que calcular. Un ángulo puede tener un valor de entre 0º a 360º, que pasado a radianes son, de 0 a 6.283 rad. Mientras que la salida, oscilará de -1.0 a 1.0.

Entonces, surge el primer imprevisto con el lenguaje de descripción de hardware, se puede trabajar con datos reales, pero los resultados simplemente son simulables, ya que una variable real no es sintetizable. Esto complica una futura implementación en una FPGA.

Para solucionarlo, se usan tipos de datos sintetizables, como son los vectores de bits, los cuales se pueden interpretar de distintas maneras que se verán más adelante. Esta interpretación se refiere a que un vector relleno de ‘1’ y ‘0’ puede contener mucha información codificada, la cual, solo hay que saber descodificar para interactuar con ella.

5.2.1. Longitud de datos almacenados

En VHDL existe un tipo estandarizado del vector de bits, denominado STD_LOGIC_VECTOR. La longitud máxima que puede alcanzar un vector es de 32 bits o 64 bits según el ancho de palabra del microprocesador donde se ejecute el software que sintetiza o simula la descripción del circuito.

En este trabajo Fin de Grado se han utilizado distintas longitudes de vectores según los datos correspondientes.

- Resultado del seno: longitudes de 8, 16 y 32 bits.
- Ángulo de entrada: longitudes de 10 y 12 bits.
- Variables intermedias: longitudes que varian entre 10 y 36 bits según sea necesario.

5.2.2. Formato de datos almacenados

Según el método utilizado para el cálclo del seno, se usa un formato diferente para representar los datos, en concreto son de dos tipos. Estos dos formatos de datos se corresponden con números enteros o números decimales.

- **Números enteros**

Un vector de bits tiene su correspondencia directa con un número entero. En este caso se han utilizado vectores de 8, 16 y 32 bits para el resultado de salida. Por tanto, se puede contar hasta 255, 65.535 y 4.294.967.295, respectivamente. Si se utilizara complemento a dos (ca2), donde el primer bit indica el signo, la cuenta sería también con números negativos, [-128,127], [-32.768, +32.767] y [-2.147.483.648, +2.147.483.647] respectivamente [16].

Un vector de bits se puede convertir en un número entero bajo la función CONV_INTEGER(variable) que proporciona la herramienta. De manera contraria se puede pasar de un número entero a un vector mediante la función CONV_STD_LOGIC_VECTOR(variable, tamaño del vector).

- **Números decimales**

Al igual que para números enteros ya hay una conversión correspondiente a binario, lo mismo sucede para números decimales. Un ejemplo de la conversión decimal a binario para el número 0.3125 sería la siguiente [17]:

0.3125 (decimal) → "0.0101" (binario)

0.3125 x 2 = 0.625 < 1 → '0'
0.6250 x 2 = 1.250 > 1 → '1' y 1.250 – 1 = 0.25
0.2500 x 2 = 0.500 < 1 → '0'
0.5000 x 2 = 1.000 = 1 → '1'

Ahora se colocan en orden los bits obtenidos, poniendo primero "0. _ _ _ _", ya que el primer número entero es cero, si fuera otro, se colocaría el número entero que le correspondiese. Finalmente, se obtiene el resultado "0.0101".

Sin embargo, este método de poder dividir un vector de bits en una parte entera y otra decimal se complica bastante a la hora de usar la herramienta de diseño de hardware utilizada. Lo que conlleva a usar un formato de almacenamiento diferente.

El siguiente método consta de una longitud de vector de 16 bits, aunque se pueden utilizar otras longitudes, la elegida es esta porque será la que de una buena precisión para el desarrollo del proyecto. El vector es dividido en posiciones positivas y negativas, por tanto, al elevar dos a la posición, se obtienen números enteros y decimales, y al sumarlos se obtiene una conversión de vector a decimal.

Vector	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**
Posición	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
Valor Pos.	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}
Nº dec.	$2^3 \cdot 0 + 2^2 \cdot 1 + 2^1 \cdot 1 + 2^0 \cdot 0 + 2^{-1} \cdot 1 + 2^{-2} \cdot 1 + 2^{-3} \cdot 1 + 2^{-4} \cdot 0 + 2^{-5} \cdot 0 + 2^{-6} \cdot 0 + 2^{-7} \cdot 0 + 2^{-8} \cdot 1 + 2^{-9} \cdot 0 + 2^{-10} \cdot 0 + 2^{-11} \cdot 0 + 2^{-12} \cdot 1 =$ **6.879150391**															

Tabla 5.1. Ejemplo conversión vector a decimal

5.2.3. Tablas para datos de entrada

- **Tablas para el diseño mediante aproximación por muestreo e interpolación lineal**

En los diseños el dato de entrada más significativo es el ángulo cuyo seno se quiere calcular. Si se trata de los dos primeros diseños, aproximación e interpolación, como se ha visto previamente, se necesita tener la señal del seno muestreada, y esta señal muestreada estará almacenada en tablas de consulta.

Según el número de muestras, habrá almacenados distintas tablas de ángulos, las cuales se pueden ver en la siguiente tabla:

Tablas	**Muestras**	**Ángulos**
Tabla_1	13	0º, 30º, 60º, 90º, 120º…360º
Tabla_2	25	0º, 15º, 30º, 45º, 60º … 360º
Tabla_3	37	0º, 10º, 20º, 30º, 40º … 360º

Tabla 5.2. Ángulos según el nº de muestras

Por ejemplo, si se toman 37 muestras del seno, lo que se hace es dividir la señal en 37 partes y estas guardarlas en las tablas. Se usa una tabla para los valores de los ángulos y otra para los valores del seno. Los valores de los ángulos van de 0º a 360º, por tanto, con 9 bits valdría. Mientras que los valores del seno oscilan entre -1.0 y +1.0, y en estos dos primeros diseños se utilizan números enteros, por tanto, hay que multiplicar por un número para hacer enteros todos los valores que oscilen en [-1,+1]. Según la longitud de bits usada para la salida, se multiplica por un número u otro, para este primer ejemplo se usan 8 bits, por tanto, hay que multiplicar por 100. Ya que con 8 bits se puede ir desde -128 a 127, y por tanto, cubre el intervalo de [-1,+1] que ahora será de [-100,+100].

Posición	**Ángulo**	**Ángulo (entero)**	**Seno**	**Seno (entero)**
0	0º	“0000000000”	0.00→0	“00000000”
10	10º	“0000001010”	0.17→17	“00010001”
20	20º	“0000010100”	0.34→34	“00100010”
30	30º	“0000011110”	0.50→50	“00110010”
…	…	…	…	…
90	90º	“0001011010”	1.00→100	“01100100”
…	…	…	…	…
360	360º	“0101101000”	0.00→0	“00000000”

Tabla 5.3. Tablas de ángulos y senos con 37 muestras (8 bits)

En la siguiente tabla se repite el proceso con 16 bits, por tanto, ahora se multiplica por 10^4. Ya que si con 16 bits se puede ir desde -32768 a +32767, al multiplicar el resultado del seno por 10^4 ahora el intervalo estará entre -10^4 a $+10^4$. En la siguiente tabla se han omitido los vectores de los ángulos porque se vuelve a hacer para 37 muestras.

Posición	Seno	Seno (entero)
0	0.0000→0	"0000000000000000"
10	0.1736→1736	"0000011011001000"
20	0.3420→3420	"0000110101011100"
30	0.5000→5000	"0001001110001000"
…	…	…
90	1.0000→10000	"0010011100010000"
…	…	…
360	0.0000→0	"0000000000000000"

Tabla 5.4. Tabla del seno con 37 muestras (16 bits)

También es importante saber que las tablas para 8 y 16 bits tienen 361 posiciones, donde solo se rellenan las muestras conocidas, es decir, en el caso de la *tabla 5.4* los valores de 1, 2, 3… están a cero, ya que nunca son utilizados.

Finalmente, el último tipo de tabla que queda es para 32 bits. Con esta cantidad de bits se puede contar desde -2.147.483.648 a +2.147.483.647, por tanto, ahora se multiplica por 10^9. Obteniéndose así un intervalo de $[-10^9,+10^9]$. En esta tabla ya no hay 361 posiciones, esta ha sido creada solo con el número de posiciones necesarias, que es el de la cantidad de muestras conocidas. Esto es debido a que el diseño con 32 bits difiere ligeramente de los de 8 y 16 bits.

Posición	Seno	Seno (entero)
0	0.000000000→0	"00000000000000000000000000000000"
10	0.173648178→173648178	"00001010010110011010100100110010"
20	0.342020143→342020143	"00010100011000101101000000101111"
30	0.500000000→500000000	"00011101110011010110010100000000"
…	…	…
90	1.000000000→1000000000	"00111011100110101100101000000000"
…	…	…
360	0.000000000→0	"00000000000000000000000000000000"

Tabla 5.5. Tabla del seno con 37 muestras (32 bits)

Vistos los diseños de las tablas con 37 muestras según la longitud de bits de la salida del seno, habrá otras tablas iguales donde lo único que cambie será el intervalo de muestreo, para 13 y 25 valores.

- **Tabla para el diseño mediante serie de Taylor**

En este otro método, no se usan los números enteros para interpretar los resultados. Se trabaja con los números decimales en el apartado anterior. Ahora las tablas de consulta van a ser diferentes. En este método, solo hace falta una tabla, esta lo único que sirve es para pasar de grados a radianes.

Para realizar la serie de Taylor es necesario que el valor de entrada a la serie esté en radianes. Este valor se puede introducir directamente como un puerto de entrada, sin embargo, es mucho más sencillo para una persona pensar directamente en grados. Además, de que es más fácil introducir un entero (grados) que un decimal (radianes) como vector de bits, tal y como se ha visto en el apartado 5.2.2.

Por tanto, se ha creado una tabla de correspondencia entre grados y radianes, donde al meter el ángulo de entrada en grados se accede a la tabla y se encuentra el valor que le pertenece en radianes. Para tener un valor del ángulo cuya aproximación llegue a los tres decimales se han necesitado usar 12 bits.

Posición	Ángulo	Ángulo (rad)	Ángulo (decimal)
0	0º	0.000	"000000000000"
1	1º	0.017	"000000000100"
2	2º	0.035	"000000001001"
3	3º	0.052	"000000001101"
…	…	…	…
60	60º	1.047	"000100001100"
…	…	…	…
90	90º	1.571	"000110010010"

Tabla 5.6. Correspondencia entre grados y radianes

Si se quisiera comprobar que los valores en binario son correctos, simplemente habría que aplicar el formato de almacenamiento de datos para número decimales explicado en el apartado 5.2.2.

También se ve que esta tabla solo va de 0 a 90, esto se debe porque para el resto de ángulos se ha establecido una relación respecto al primer cuadrante, tal y como se ha visto en el capítulo 2, obteniéndose así el resultado válido.

5.3. Diagramas de flujo

Los circuitos digitales no se realizan mediante flujogramas, lo que se hace es definir las rutas de datos que van a seguir estos y se trazan los cronogramas esperados. Sin embargo, para entender cual va a ser el proceso que va a realizar cada diseño, se explican brevemente varios diagramas de flujo de los diseños, aunque estos estén relacionados con la descripción de software, sirven para una mejor compresión de lo que se está realizando. En los siguientes apartados, se explicarán detalladamente las rutas de datos de cada diseño.

5.3.1. Diagrama para la aproximación e interpolación

- **Longitud de 8 y 16 bits**

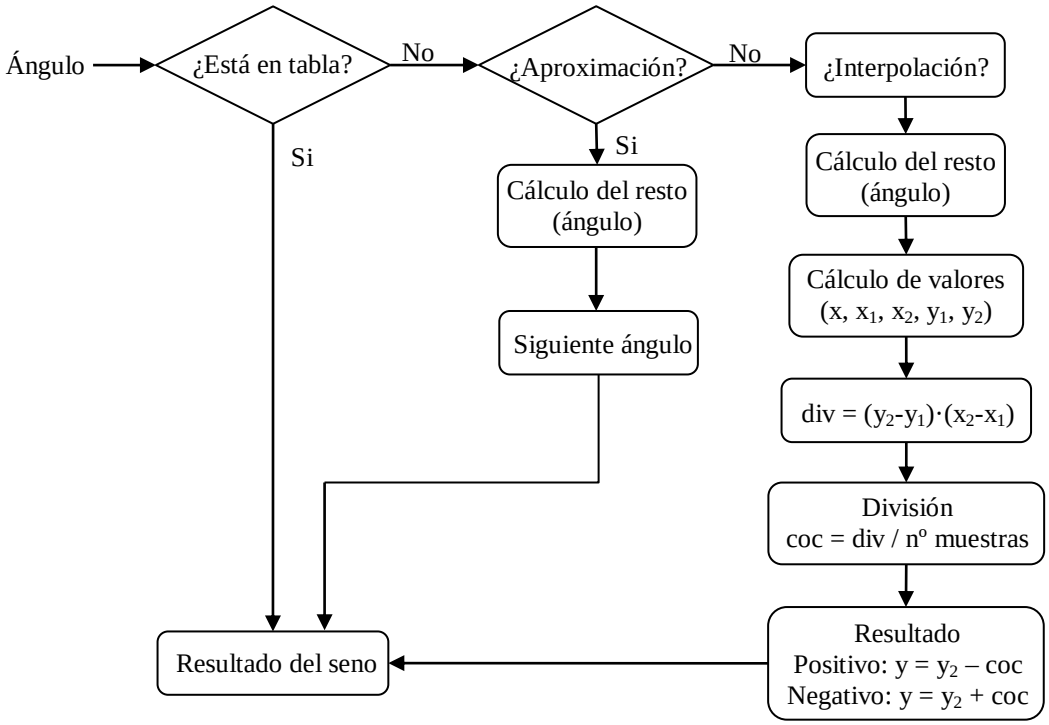


Figura 5.1. Diagrama del funcionamiento del diseño por aproximación e interpolación (8 y 16 bits)

En el anterior diagrama se pueden apreciar tres partes diferentes, como son la de la tabla, la de aproximación y la de interpolación. A continuación se verá que se realiza en cada parte.

Lo primero es introducir el valor del ángulo, se busca en la tabla y si es encontrado en esta, se saca directamente su valor.

Si el valor no ha sido encontrado en la tabla, se mira si se ha elegido la opción de aproximar su valor. Si es así, se calcula el resto del ángulo introducido. Es decir, si el ángulo es el 37º y se conocen los ángulos de 30º y 40º (esto sería para 37 muestras) se calcula el resto de 37 entre 10. El cual da 7. Entonces se halla cuál es el siguiente ángulo conocido: *siguiente ángulo = 37 + (10 – 7) = 37 + 3 = 40.* Una vez conocido el siguiente ángulo, se saca el valor de su seno accediendo a la tabla. Así se consigue el valor del seno de 37º aproximándolo al de 40º.

Si no se ha elegido la opción de aproximar el valor, se ha elegido la opción de interpolarlo. Con lo cual se vuelve a calcular el resto del ángulo igual que antes, y una vez hecho esto, se conocen los valores de x, x_1, x_2, y_1, y_2, pertenecientes a la fórmula de interpolación, explicada en el apartado 2.3.2. Una vez hallados los valores, se aplica la fórmula teniendo en cuenta si el valor de la interpolación se está haciendo para valores positivos o negativos, obteniendo así el resultado final del seno.

- **Longitud de 32 bits**

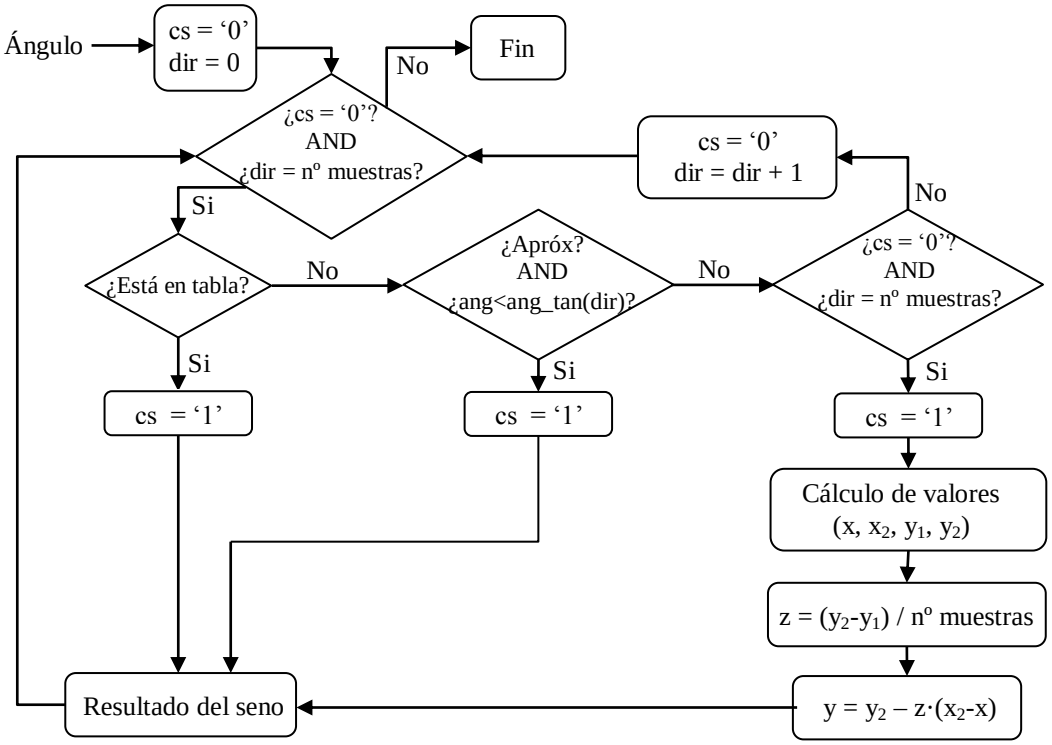


Figura 5.2. Diagrama del funcionamiento del diseño por aproximación e interpolación (32 bits)

Este otro diagrama está gobernado bajo un bucle WHILE el cual pregunta si *cs* está activado, entonces es que ya se ha obtenido el valor del seno independientemente del método, y además pregunta si se ha llegado hasta el número total de muestras. Si no se han producido estas dos cosas a la vez entonces es que no se ha obtenido un valor para el seno.

Después del bucle aparecen de nuevo tres ramales, uno para el acceso directo a la tabla, otro para la aproximación al siguiente seno y el último, que es el de la interpolación. En los dos primeros, la obtención de resultados es directa. Sin embargo, en el último hay que aplicar la fórmula de interpolación. En este caso además, existe un paso intermedio *(z)*, porque como al haber 32 bits, si se hace directamente se desbordan algunos datos. Por ello, el paso intermedio consiste en realizar primero la división y a posteriori, la multiplicación.

Si finalmente no se entra en ningún ramal, se pasa al siguiente ángulo de la tabla, y se vuelve a entrar al bucle. Realizando de nuevo las comparaciones necesarias entre el ángulo cuyo seno se quiere calcular y el de tabla.

Este diseño realmente lo que muestra, es un recorrido por la tabla desde el ángulo de 0º hasta el ángulo de la tabla inmediatamente superior al de entrada. Por ejemplo, si se mete a calcular el seno de 37º, el diseño hará un recorrido estancándose en 40º, bajo el cual opera y realiza la aproximación o interpolación, según haya sido indicado.

5.3.2. Diagrama para la Serie de Taylor

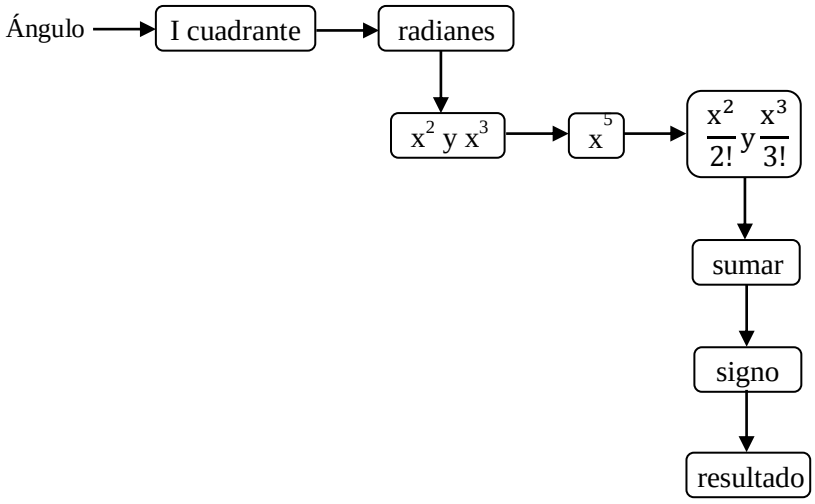


Figura 5.3. Diagrama del funcionamiento del diseño mediante la serie de Taylor

Este diseño es mucho más complejo que el anterior. El diagrama muestra simplemente una idea principal, sin embargo, en el apartado 5.5 se ve un estudio más detallado. Lo primero es conseguir relacionar el ángulo de entrada con otro del primer cuadrante y pasarlo a radianes. Después se hallan los términos de x^3 y x^5, necesarios en la serie de Taylor, y posteriormente, se dividen para obtener los términos correspondientes a la serie. Finalmente se suman los términos y se estudia el signo, obteniendo así el resultado final.

5.4. El seno mediante aproximación por muestreo y por interpolación lineal

En los siguientes apartados se detalla la arquitectura del diseño, la ruta de datos y posteriormente la descripción en VHDL. Todo ello en función de las longitudes que se estén usando en cada momento.

5.4.1. Introducción

Como ya se ha visto anteriormente, el seno se realiza a partir de un número de muestras tomadas previamente, las cuales han sido almacenadas en tablas. Dentro de la descripción lo único que se hace es elegir al principio el número de muestras que se quieren. Podrán ser 13, 25 o 37, por tanto, según se elija un valor u otro, se accede a los valores de una tabla u otra, y a partir de ahí, comienza el desarrollo del diseño.

El almacenamiento de datos se hace mediante números enteros. También hay que saber la diferencia existente en los distintos errores que se están cometiendo. Un error es el propio cálculo del seno mediante aproximaciones, donde si se introduce el ángulo de 36º se saca en realidad el valor de 60º, 45º ó 40º dependiendo del nº de muestras usadas. Y el otro error cometido es la cantidad de decimales de precisión

usados, donde según se aumenta la longitud de vector, también aumenta el número de decimales de precisión.

En el caso de la aproximación, la descripción de hardware llevada a cabo es muy sencilla, ya que simplemente es evaluar cual es el ángulo de entrada, si está en la tabla correspondiente, se saca el valor del seno asignado, y si no está como tal en la tabla, simplemente sacar el valor del seno siguiente. Al ser tan sencilla, se ha decidido implementar de forma conjunta el diseño por aproximación e interpolación. El método de interpolación matemáticamente no es complicado, sin embargo, para la implementación hardware se han tenido ciertas dificultades, las cuales se resuelven detalladamente en los siguientes apartados.

5.4.2. Longitudes de 8 y 16 bits

La distinción hecha a la hora de utilizar distintas longitudes de vector es la de su posible sintetización, es decir, la posibilidad de llevar a cabo la implantación del diseño en una FPGA. Para 8 y 16 bits si es sintetizable el diseño, mientras que para 32 bits solo es simulable. Debido a esto, el diseño es diferente.

La gran dificultad encontrada para el desarrollo de este diseño está en la división, la cual no es sintetizable. Se recuerda la fórmula explicada en el capítulo 2 bajo la cual se desarrolla el diseño:

$$y = y_2 - \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_2 - x)$$

Donde las *x* indican ángulos y las *y* indican senos. Para el diseño se conocen todos los valores excepto el de *y*, que es el resultado del seno. El problema aparece al resolver la división de x_2-x_1.

Dentro del diseño hardware la división no es una operación sencilla, y su implementación requiere de algoritmos bastante complejos. Solamente se puede dividir por potencias de dos, ya que lo que se esta haciendo es un desplazamiento de bits. Es decir, dividir por 2^n significa desplazar *n* posiciones el vector a la derecha, mientras que multiplicar por 2^n será desplazar *n* posiciones el vector a la izquierda. Nuestra división no es por potencias de dos, sin embargo, sí que es siempre por un valor fijo. Este valor fijo depende del número de muestras elegido. En la siguiente tabla se ve rápidamente los valores que puede tomar el dividendo de esta división:

Muestras	**Ángulos**	x_2-x_1
13	0º, 30º, 60º, 90º…	30
25	0º, 15º, 30º, 45º…	15
37	0º, 10º, 20º, 30º…	10

Tabla 5.7. Valores del dividendo x_2-x_1

Por tanto, teniendo el valor del divisor fijo y sabiendo que la división es de números enteros, se ha realizado una solución particular. Esta división simplemente consta de realizar una serie de iteraciones mediante un FOR, en el cual se resta el valor

del divisor al dividendo, y se cuenta el número de iteraciones, las cuales definen el cociente. Por ejemplo, realizamos 50 entre 10:

```
dividendo := 50;
FOR i IN 0 TO 6 LOOP
       IF (dividendo >= 10) THEN
              cociente  := cociente + 1;
              dividendo := dividend – 10;
       END IF;
END LOOP;
```

Figura 5.4. División con bucle FOR

Este tipo de división ha sido necesaria utilizarla en alguna parte más del diseño, debido a la constante utilización de números enteros.

5.4.3. Diseño con 8 y 16 bits

En este apartado hay que destacar tres partes fundamentales: la arquitectura del diseño, la ruta de datos y la descripción en VHDL. Todas ellas se van viendo a continuación.

- **Arquitectura**

En la siguiente figura se puede apreciar la arquitectura llevada a cabo para el diseño y el control de datos.

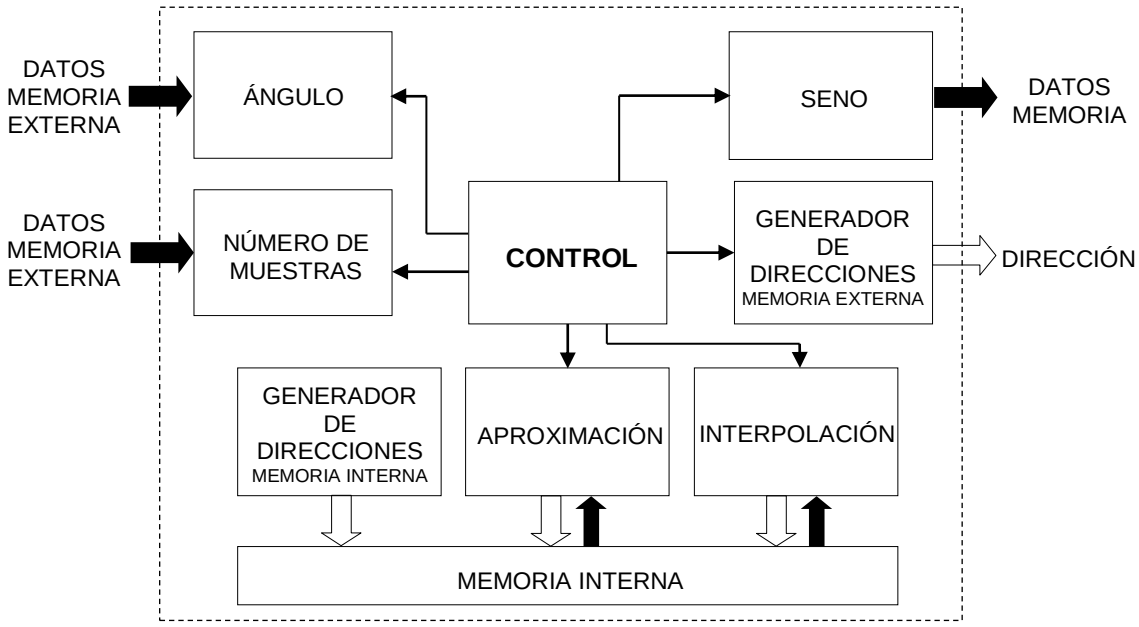


Figura 5.5. Arquitectura del diseño por aproximación e interpolación para 8 y 16 bits

El bloque central se encarga del control del resto de bloques. Cada bloque tiene una función tal y como se puede apreciar. Los bloques de aproximación e interpolación son el epicentro de este diseño, aunque estén juntos en el diseño, aparecen por separado porque se ejecuta uno u otro, pero nunca los dos a la vez.

- **Ruta de datos**

La ruta de datos se describe en la siguiente figura. Esta hecho para 8 bits, para 16 bits es exactamente igual, lo único que cambiarían son los valores de las divisiones y la multiplicación. Además, de las longitudes de los datos.

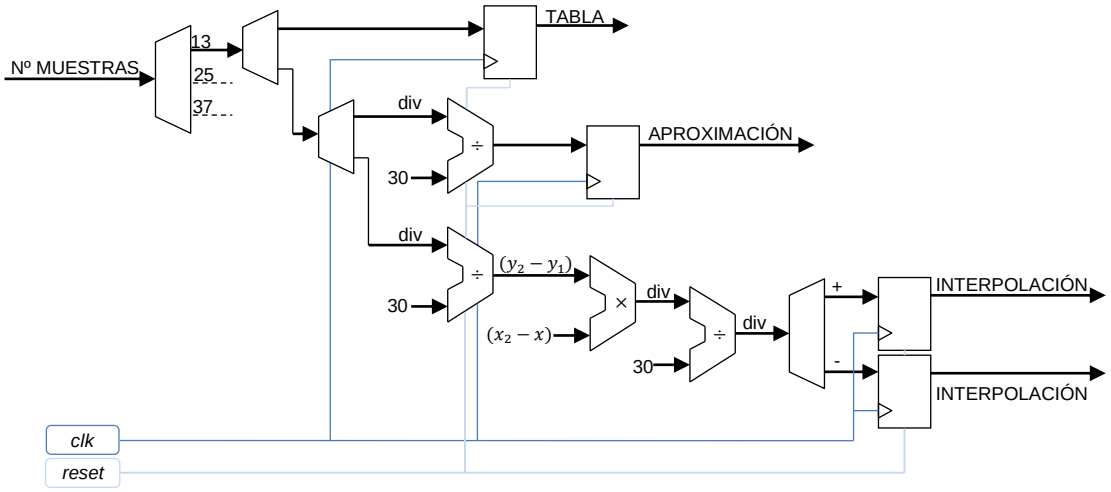


Figura 5.6. Ruta de datos del diseño por aproximación e interpolación para 8 bits

- **Descripción en VHDL**

En las siguientes líneas se detallan la arquitectura y la ruta de datos a nivel de descripción VHDL. La explicación dada corresponde al código de 8 bits. Para 16 bits sigue el mismo mecanismo, lo único que cambia son los tamaños de variables, de los bucles necesarios, etc.

En primer lugar se añaden las bibliotecas necesarias, los paquetes que contienen las tablas con las muestras, y a continuación, se diseña la entidad.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

--Tablas para almacenar los valores del seno conocidos (13,25,37 muestras)
use work.tabla_13.all;
use work.tabla_25.all;
use work.tabla_37.all;

entity seno is
 port( clk       : in std_logic;
       reset     : in std_logic;
       angulo    : in std_logic_vector (9 downto 0);
       tamaño    : in std_logic_vector (6 downto 0);  --13, 25 o 37 muestras
       bus_datos : out std_logic_vector (7 downto 0); --Valor del seno
       eleccion  : in std_logic); -- '0': Aproximacion, '1':Interpolacion
```

Figura 5.7. Bibliotecas, paquetes y entidad. Diseño aproximación e interpolación (8 bits).

Posteriormente se crea la arquitectura de la entidad. Lo primero es crear un PROCESS cuya lista de sensibilidad tenga un *reset* y un *clock*. Se crean las variables intermedias necesarias y se abre un CASE, que sirve para elegir el número de muestras.

```
architecture funcion of seno is
begin
 process(clk,reset)
   variable bus_dir         : integer range 0 to 361;
   variable y2,y1,y,x2,x1,x : integer; --Puntos para la interpolación
   variable div,coc,i,j     : integer; --Parametros para la interpolacion
 begin
   if reset = '1' then
      bus_datos <= "00000000";
   elsif clk'event and clk = '1' then
      bus_dir := conv_integer (angulo); --Ángulo: binario a entero
      case tamaño is
         when "0001101" => (…)    --13 muestras
         when "0011001" => (…)    --25 muestras
         when "0100101" => (…)    --37 muestras
      end case;
   end if;
 end process;
end funcion;
```

Figura 5.8. Arquitectura en VHDL. Diseño aproximación e interpolación (8 bits).

Comienza a hacerse el diseño para el primer caso, el de 13 muestras. Para el resto de posibilidades se hace igual, lo único que se accedería a una tabla diferente, las divisiones tendrían otros valores, etc. pero el mecanismo sería el mismo.

Las dos primeras partes constan del acceso directo a la tabla y de la aproximación, que serían los dos primeros ramales de la *figura 5.10*.

```
if(angulo = angulos_13(bus_dir)) then  --ACCESO DIRECTO A LA TABLA
   bus_datos <= senos_13(bus_dir);
elsif(eleccion = '0') then             --APROXIMACIÓN AL SIGUIENTE
   div := bus_dir;
   for i in 0 to 12 loop               --Se calcula el resto del ángulo
      if (div > 29) then
         div := div - 30;
      end if;
   end loop;
   y2 := bus_dir + (30 - div);         --Se halla el ángulo siguiente
   bus_datos <= senos_13(y2);          --Resultado del seno
```

Figura 5.9. Acceso a tabla y aproximación. Diseño aproximación e interpolación (8 bits).

La última parte es la de la interpolación, la cual es más larga, debido a que es más complicada de realizar.

```
elsif(eleccion = '1') then             --INTERPOLACIÓN LINEAL
   div := bus_dir;
   for i in 0 to 12 loop               --Se calcula el resto del ángulo
      if (div > 29) then
         div := div - 30;
      end if;
   end loop;
```

```
   x  := bus_dir;                        --Valor del ángulo
   x1 := bus_dir - div;                  --Valor del ángulo anterior
   x2 := bus_dir + (30 - div);           --Valor del ángulo siguiente
   y1 := conv_integer(senos_13(x1));     --Valor del seno anterior
   y2 := conv_integer(senos_13(x2));     --Valor del seno siguiente

   div := (y2-y1)*(x2-x);   --La fórmula para interpolar es:
   coc := 0;                -- y = y2 - (((y2-y1)/(x2-x1))*(x2-x))
   i   := 0;                --(fórmula para valores del seno positivos)
                            --Primero se ha hecho la multiplicación
   for i in 0 to 50 loop    --Después hacemos la división:
      if(div > 30) then     --División para valores positivos
         coc := coc + 1;
         div := div - 30;
      elsif(div < -30) then --División para valores negativos
         coc := coc + 1;
         div := div + 30;
      end if;
   end loop;

   if (div >= 0 and div <= 30) then     --Resultado para valores positivos
      y := y2 - coc;
   elsif (div >= -30 and div < 0) then  --Resultado para valores negativos
      y := y2 + coc;
   end if;

   bus_datos <= conv_std_logic_vector(y,8); --Resultado del seno
end if;
```

Figura 5.10. Interpolación. Diseño aproximación e interpolación (8 bits).

5.4.4. Longitud de 32 bits

Tal y como se había anunciado ya en el apartado 5.4.3, la diferencia que hay entre las anteriores longitudes frente a esta, es que con 32 bits no es sintetizable. Esto se debe a que a la hora de realizar las divisiones pertinentes mediante los bucles FOR, estas son demasiado lentas, debido a que ahora la longitud del vector es mucho mayor, lo que conlleva a tener números enteros muy elevados, consecuencia directa de que los bucles sean tan largos. Por tanto, simplemente se ha realizado un diseño simulable para estudiar el error que se comete con una longitud de 32 bits, y saber si realmente hay una diferencia notable frente a las otras longitudes.

5.4.5. Diseño con 32 bits

Al igual que para otras longitudes, en este apartado se explica la arquitectura, la ruta de datos y la descripción hardware llevada a cabo.

- **Arquitectura**

La arquitectura empleada para esta longitud es igual que para las anteriores longitudes de vector, simplemente cambia el control del proceso y las rutas de datos, que se verán justo después.

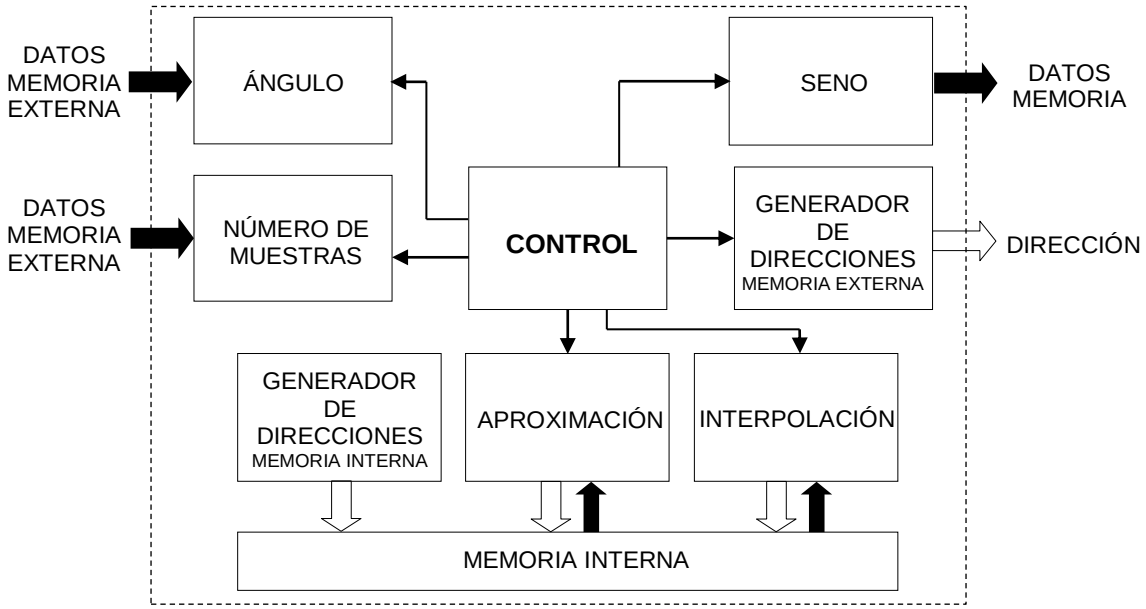


Figura 5.11. Arquitectura del diseño por la aproximación e interpolación para 32 bits

- **Ruta de datos**

A continuación se muestra la ruta de datos diseñada. Se puede apreciar que el número de divisiones ha disminuido, y las operaciones que aparecen, son más sencillas. Aparece una retroalimentación, constituida por el bucle WHILE, ya que este es el gran cambio en el diseño respecto al anterior.

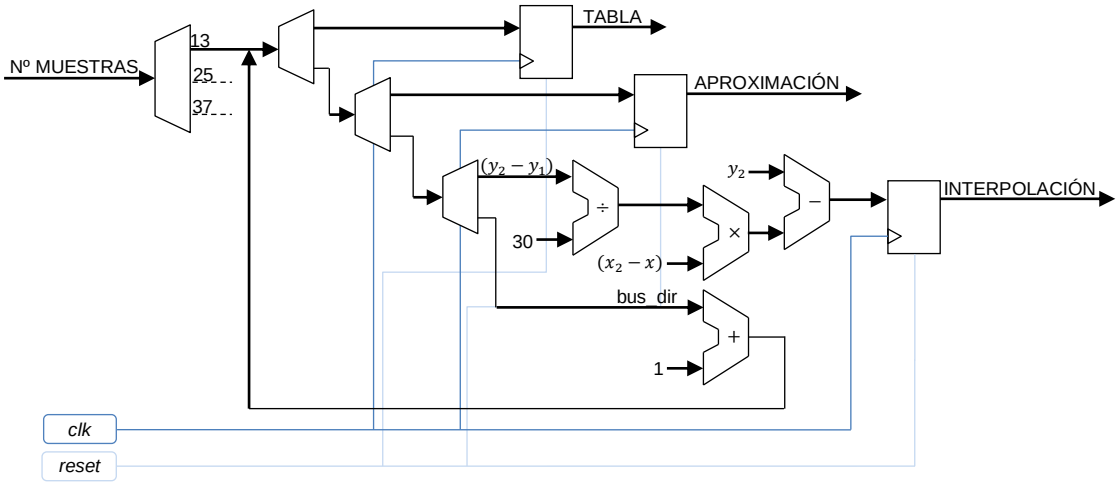


Figura 5.12. Ruta de datos del diseño por aproximación e interpolación para 32 bits

- **Descripción en VHDL**

Durante el apartado se ha visto que la funcionalidad sigue siendo la misma, incluso el diseño es similar, solamente hay pequeñas modificaciones.

Las tres primeras partes vuelven a ser las librerías, los paquetes y la entidad. En este caso se ha utilizado un único paquete donde guardamos todas las tablas necesarias.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

--Tablas para almacenar los valores del seno conocidos
use work.tabla.all;

entity seno is
 port( clk       : in std_logic;
       reset     : in std_logic;
       angulo    : in std_logic_vector (9 downto 0);
       tamaño    : in std_logic_vector (6 downto 0);   --13,25,37 muestras
       bus_datos : out std_logic_vector (31 downto 0); --Valor del seno
       eleccion  : in std_logic); -- '0': Aproximacion, '1':Interpolacion
```

Figura 5.13. Bibliotecas, paquetes y entidad. Diseño aproximación e interpolación (32 bits).

De nuevo, el siguiente paso es describir la arquitectura, el PROCESS correspondiente y un CASE para el elegir el número de muestras deseado.

```
architecture funcion of seno is
begin
 process(clk,reset)
   variable bus_dir: integer;         --Ángulo
   variable cs: std_logic;            --Flag para salir del while
   variable tamaño2: integer;         --N° de muestras
   variable y2,y1,y,x2,x: integer;    --Puntos para la interpolación
   variable z: integer;               --Parametros para la interpolacion
 begin
   tamaño2 := conv_integer(tamaño);
   if reset = '1' then
      bus_datos <= "00000000";
   elsif clk'event and clk = '1' then
      bus_dir := conv_integer (angulo);--Ángulo(bin) a entero (comodidad)
      case tamaño2 is
         when (13) =>       (…)    --13 muestras
         when (25) =>       (…)    --25 muestras
         when (37) =>       (…)    --37 muestras
      end case;
   end if;
 end process;
end funcion;
```

Figura 5.14. Arquitectura en VHDL. Diseño aproximación e interpolación (32 bits).

Igual que antes, se describe uno de los casos, de nuevo el de 13 muestras. En este caso, al cambiar el diseño, se ha introducido un WHILE, el cual depende de dos factores. Uno de ellos es *cs*, que indica si se tiene ya o no un valor del seno, y el otro es *bus_dir* que indica cuando se ha llegado al final del número de muestras disponibles. Por tanto, no se sale del WHILE hasta que no halla un resultado del seno y no se haya llegado al final del número de muestras.

En la siguiente figura se muestra simplemente el acceso directo a la tabla de 13 muestras y la aproximación correspondiente.

```
bus_dir := 0;       --Inicializo valores antes del while
cs := '0';

while (cs = '0' and bus_dir <= 13) loop
   if(angulo = angulos_13(bus_dir)) then                      --ACCESO TABLA
      cs := '1';
      bus_datos <= senos_13(bus_dir);
   elsif(eleccion = '0' and angulo < angulos_13(bus_dir))then --APROXIMAR
      cs := '1';
      bus_datos <= senos_13(bus_dir);
```

Figura 5.15. Acceso a tabla y aproximación. Diseño aproximación e interpolación (32 bits).

Posteriormente viene el diseño de la interpolación, que vuelve a ser igual, lo único que ahora se hace dividiendo directamente. Al estar trabajando con 32 bits, si se hace la operación entera de manera directa surge el desbordamiento de algunos datos, por tanto, se introduce un paso intermedio para evitarlo. Este paso simplemente es realizar primero la división correspondiente, y luego la multiplicación.

```
 elsif(eleccion = '1' and angulo < angulos_13(bus_dir)) then  --INTERPOLAR
   cs := '1';
   y2 := conv_integer(senos_13(bus_dir));      --Seno siguiente
   y1 := conv_integer(senos_13(bus_dir-1));    --Seno anterior
   x2 := conv_integer(angulos_13(bus_dir));    --Angulo siguiente
   x := conv_integer(angulo);                  --Angulo

--La fórmula para interpolar es: y = y2 - (((y2-y1)/(x2-x1))*(x2-x))
   z := (y2-y1)/30;        --Primero dividimos: (x2-x1)=30
   y := y2 - z*(x2-x);     --Despues multiplicamos (para evitar desbordar)
   bus_datos <= conv_std_logic_vector(y,32);  --Resultado del seno
 else                                         --NINGUNA CONDICIÓN
   cs := '0';
   bus_dir := bus_dir + 1;                    --Pasamos al siguiente ang.
 end if;
end loop;
```

Figura 5.16. Interpolación. Diseño aproximación e interpolación (32 bits).

Lo último que queda por matizar es el último ELSE. Este sirve para que si no se ha cumplido ninguna condición se pase al ángulo siguiente de la tabla. Realmente lo que está sucediendo en este diseño es un recorrido por todos los valores de la tabla, desde 0º hasta el ángulo inmediatamente superior al de entrada. Una vez se llega a este ángulo, se aplica la operación elegida para calcular el seno.

5.5. El seno mediante la serie de Taylor

Durante este apartado se volverán a describir las tres partes fundamentales del diseño, como son, la arquitectura, la ruta de datos y la descripción en VHDL de la función seno mediante la serie de Taylor. Este método va a ser el más óptimo de todos los estudiados.

En este diseño se utilizan vectores de bits, cuyo formato de almacenamiento es en decimales, tal cuál ha sido explicado en el apartado 5.2.2. Los ángulos de entrada cuentan con vectores de 12 bits, cuyas posiciones van de la 3 hasta la -8. Mientras que el bus de datos de salida que almacena el valor del seno esta formado por 16 bits, y sus posiciones van de la 3 hasta la -12.

5.5.1. Operaciones en formato decimal

Las operaciones que se pueden realizar con vectores son independientes del valor que puedan almacenar. Es decir, se puede dotar a un vector de bits cualquier tipo de codificación, que tras operar con otros vectores y hallar el resultado, este seguirá estando correctamente codificado.

El siguiente ejemplo muestra la suma de dos vectores codificados mediante números enteros y números decimales. Las codificaciones son las explicadas ya en apartados anteriores. Simplemente decir que para simplificar el proceso se ha disminuido el número de bits. Ahora hay 8 bits, para el formato decimal serán 4 bits para la parte entera, y los otros cuatro para la parte decimal [18].

Binario		Dec.		Ent.
0 1 0 1 0 0 1 0	-	5.125	-	82
+ 0 0 1 1 1 0 0 0	-	3.500	-	56
1 0 0 0 1 0 1 0	-	8.625	-	138

Figura 5.17. Suma binaria

En la anterior figura se aprecia la equivalencia de los vectores en ambos formatos. Se hace la suma binaria, y se obtiene un dato del cual se extrae su valor según las codificaciones.

Sin embargo, para nuestro método de codificación decimal, esto no va a ser siempre tan fácil. El problema viene cuando aparecen otro tipo de operaciones. En este proyecto, la operación principal es la multiplicación. Al hacer una multiplicación, esa coma fija se ve desplazada, por tanto, para poder interpretar un resultado correcto tras la multiplicación, esta coma debe ser recolocada.

Binario		Dec.		Ent.
0 0 1 0 1 0 1 0	-	2.6250	-	42
x 0 0 1 0 1 0 1 0	-	2.6250	-	42
1 1 0 1 1 1 0 0 1 0 0	-	13.7812	-	1764

Figura 5.18. Multiplicación binaria I

Se puede apreciar en el anterior ejemplo que al interpretar el valor directamente después de la multiplicación, la codificación extraída es incorrecta para el formato decimal. Teniendo en cuenta que los cuatro primeros bits de la izquierda pertenecen siempre a la parte entera del número en base decimal.

Binario		Dec.		Ent.
0 0 1 0 1 0 1 0	-	2.6250	-	42
x 0 0 1 0 1 0 1 0	-	2.6250	-	42
0 1 1 0 1 1 1 0 0 1 0 0	-	6.8906	-	1764

Figura 5.19. Multiplicación binaria II

Sin embargo, si se recoloca el resultado obtenido sí que se extrae el valor correcto, tal y como se puede apreciar en la imagen anterior. Para realizar esta reestructura simplemente se ha metido un cero por la izquierda. La manera de recolocar

es diferente según los valores de la multiplicación. En nuestro diseño, se ha creado una función la cual se encargará de este reordenamiento en función de los valores que se tengan.

5.5.2. Segmentación

La segmentación es un proceso que se realiza para aumentar la velocidad de operación del diseño. Consiste en hacer un registro coordinado de los datos para usarlos cuando sea debido [19]. Es cierto que la segmentación conlleva una latencia inicial, que retrasa el primer resultado a obtener. Sin embargo, gracias a esto, los siguientes resultados que se quieran obtener tardan mucho menos.

Con una función sencilla se explica el proceso de segmentación, según segmentemos más, menos o nada. Se parte de la siguiente función: $y = Ax_0 + Bx_1$:

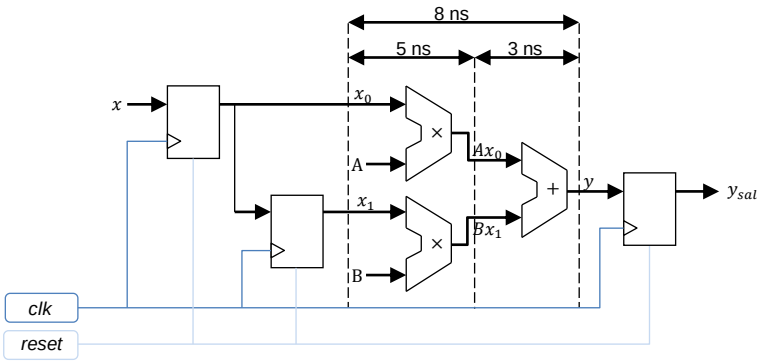


Figura 5.20. Diseño 1 con segmentación

En este ejemplo se ve que el camino crítico es el que está entre ambos registros, que sería de 8 ns. Lo que conlleva a que cada ciclo de reloj fuese mínimo de 8 ns. Que es lo que tarda en sacar un resultado desde que hay un dato de entrada. Estos tiempos no son propiamente reales, ya que dependen de la tecnología usada en los bloques, simplemente son ejemplos para explicar la segmentación. En el cronograma también se puede ver la latencia inicial de la que se hablaba.

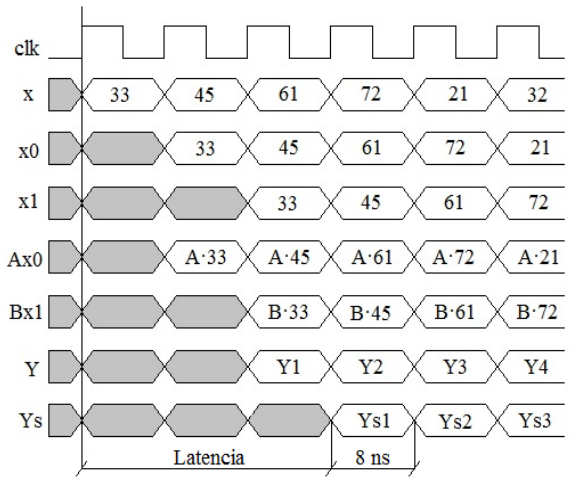


Figura 5.21. Cronograma del diseño 1

Existen tres ciclos de reloj de latencia, o lo que es lo mismo, 24 ns. Esta latencia se debe a la segmentación. Lo que conlleva es que tarda más en llegar el primer resultado, sin embargo, el resto tardan mucho menos, solo un ciclo de reloj. La gran ventaja de la segmentación es acortar los tiempos de obtención de resultados para cuando se quiera hacer el seno de muchos datos seguidos.

Como se ha visto, esta sería una forma de segmentar el diseño, sin embargo, hay otras, mediante las cuales se puede intentar bajar el camino crítico en detrimento de la latencia. Para bajar el retardo, lo que se hace es introducir registros intermedios.

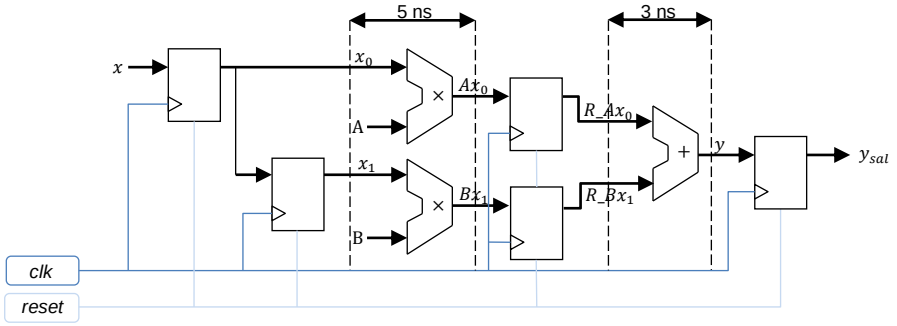


Figura 5.22. Diseño 2 con segmentación

Puede verse como al introducir un registro nuevo, el camino crítico existente entre biestables es de 5 ns. Por tanto, se ha bajado el tiempo de ciclo. Sin embargo, al meter un biestable más, el tiempo de latencia ha aumentado, tal y como se ve en el cronograma.

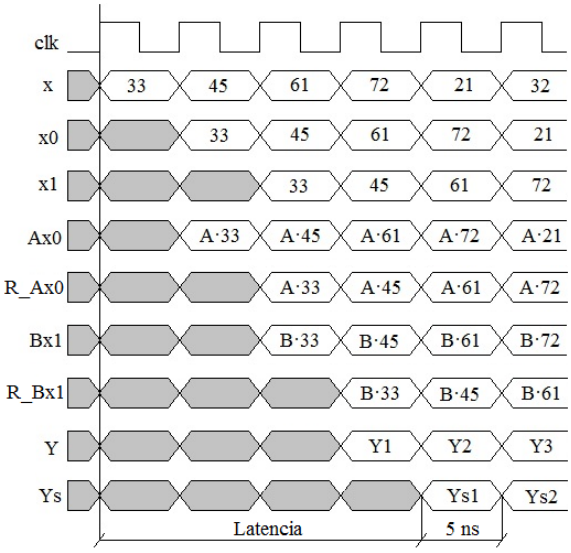


Figura 5.23. Cronograma del diseño 2

Para explicar la segmentación se ha usado un ejemplo sencillo. Sin embargo, en nuestro diseño ha habido que tener muchos más factores en cuenta. Se ha buscado bajar el retardo en el camino más crítico, el cual se aprecia en la *figura 5.24*. También ha habido que ir registrando algunas señales adecuadamente para poder usarlas cuando era necesario. Ya que hay que realizar las operaciones de acuerdo al ángulo de entrada, por tanto, cuando haya que sumar todos los términos de la serie de Taylor, estos deben estar referidos al mismo ángulo, ya que si no el resultado será erróneo.

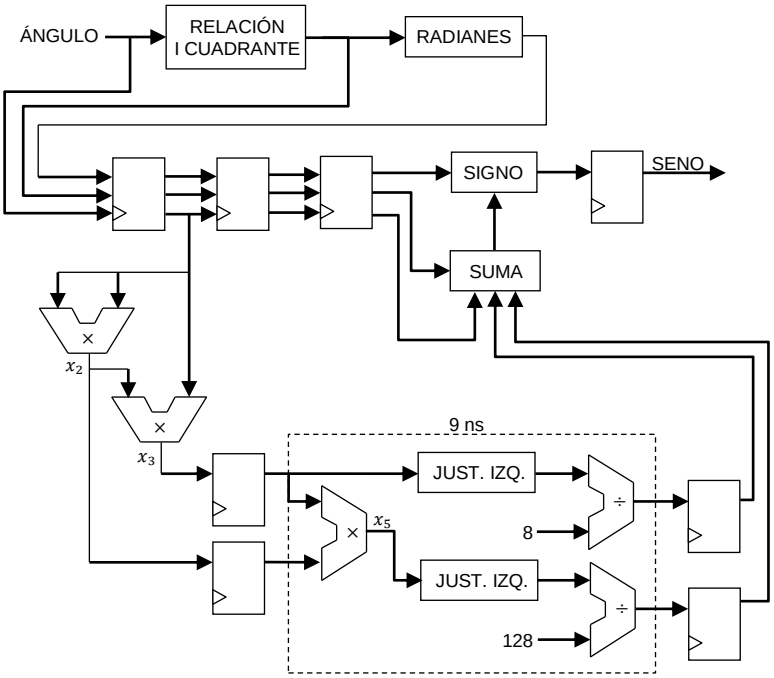


Figura 5.24. Segmentación del diseño de la función seno mediante la Serie de Taylor

Simplemente viendo el esquemático general, ya que ni siquiera es el RTL detallado, se aprecia que el diseño es bastante más complejo. Entre los biestables que registran los valores de x^2 y x^3, y los que registran las divisiones de los términos x^3 y x^5, se encuentra el camino más crítico. También se debe a la aparición de la función *Just. Izq.* que se verá posteriormente con mayor detalle.

Finalmente se realiza el cronograma correspondiente a nuestro diseño.

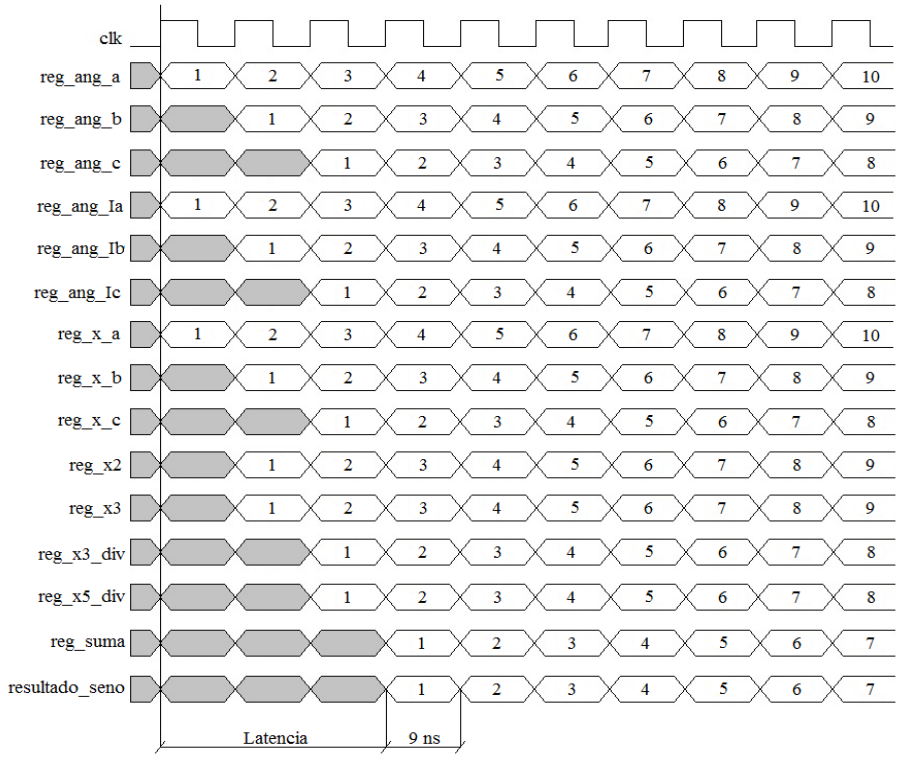


Figura 5.25. Cronograma del diseño de la función seno mediante la Serie de TAlor

En este caso la latencia es de tres ciclos, concretamente de 27ns. El ciclo es de 9ns, tal y como se indicó anteriormente. Hay que recordar que los tiempos de retardo de los bloques no son fijos, sino que varian dependiendo de la tecnología utilizada.

Como se dijo al principio, estos tiempos no son los reales de nuestros diseños. En el trajo Fin de Grado hay un análisis completo de la velocidad de operación de los diseños, que corresponde al capítulo 7.

5.5.3. Diseño

Una vez vistos estos conceptos previos, se pasa a realizar el diseño de la función seno mediante la serie de Taylor. Para ello volvemos a explicar las tres partes fundamentales del diseño.

- **Arquitectura**

La arquitectura empleada para este diseño es totalmente diferente a las anteriores. Ahora el bloque principal es el de serie de Taylor, el cual realiza la función seno, y también aparece otro bloque denominado funciones, que contiene dos funciones fundamentales para la realización del seno.

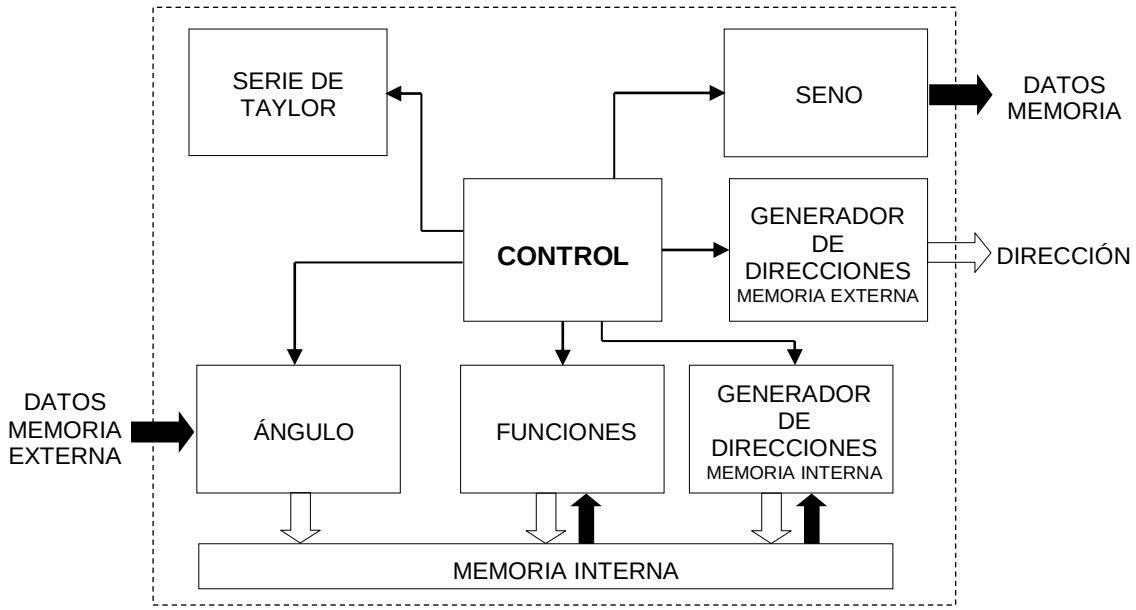


Figura 5.26. Arquitectura del diseño mediante la serie de Taylor

- **Ruta de datos**

En la siguiente figura se muestra la ruta de datos del bloque serie de Taylor de la *figura 5.27*.

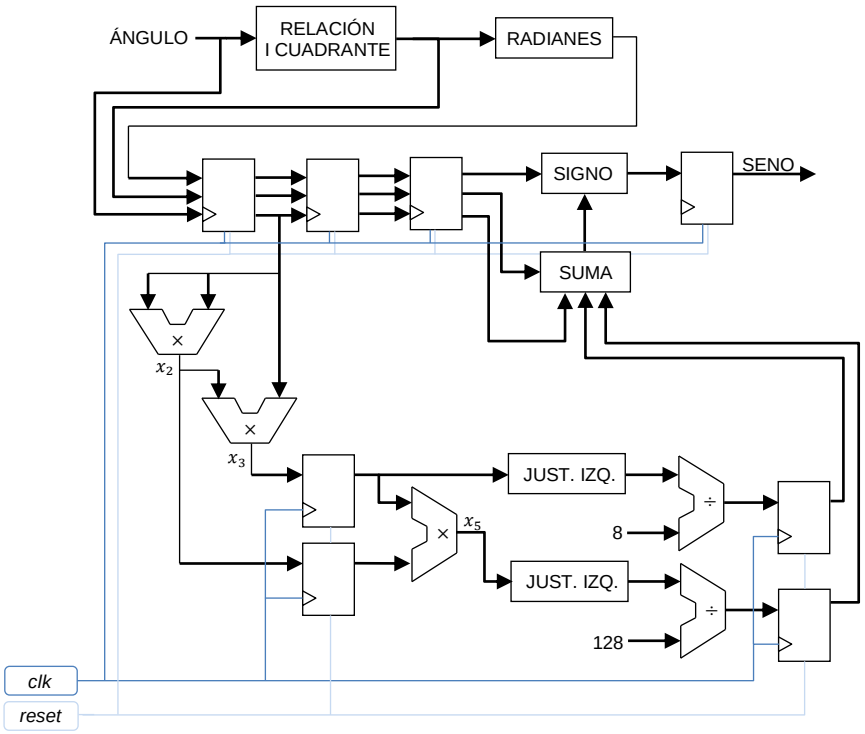


Figura 5.27. Ruta de datos del diseño mediante la serie de Taylor

- **Descripción en VHDL**

Al igual que en los anteriores diseños, lo primero es introducir las librerías necesarias, los paquetes y crear la entidad. En este caso aparece un paquete con la tabla que pasa de grados a radianes y otro con las funciones necesarias. Ahora la entidad es muy sencilla, ya que a parte del *clock* y del *reset*, solo va a estar el ángulo de entrada y la salida del seno.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

use work.tabla.all;        --Tabla que pasa de grados a radianes
use work.funciones.all;    --Funciones descritas

entity taylor is
 port( clk       : in std_logic;
       reset     : in std_logic;
       angulo    : in std_logic_vector (9 downto 0);   --Angulo
       bus_datos : out std_logic_vector (15 downto 0); --Valor del seno
end taylor;
```

Figura 5.28. Bibliotecas, paquetes y entidad. Diseño mediante la serie de Taylor.

El siguiente paso vuelve a ser crear la arquitectura. Ahora aparecen muchas más señales intermedias, debido a la gran cantidad de registros que hay que hacer. Las señales que lleven *a, b, c* al final, significan que van registradas 0, 1 y 2 ciclos respectivamente.

```
architecture Behavioral of taylor is
--Señales para registrar el ángulo verdadero que entra (0-360)
   signal reg_bus_dir_a : integer := 0;
   signal reg_bus_dir_b : integer := 0;
   signal reg_bus_dir_c : integer := 0;

--Señales para registrar el ángulo con el que presentaria relacion en el
--primer cuadrante
   signal reg_bus_dir_Ia : integer := 0;
   signal reg_bus_dir_Ib : integer := 0;
   signal reg_bus_dir_Ic : integer := 0;

--Señales para registrar el ángulo del I cuadrante en radianes
   signal reg_xa : std_logic_vector (11 downto 0) := "000000000000";
   signal reg_xb : std_logic_vector (11 downto 0) := "000000000000";
   signal reg_xc : std_logic_vector (11 downto 0) := "000000000000";

--Señales para registrar el valor de x^2 y x^3
   signal reg_x2_int: std_logic_vector (23 downto 0) := "000000000000 … 0";
   signal reg_x3_int: std_logic_vector (35 downto 0) := "000000000000 … 0";

--Señales para registrar el valor de x^3/8 y x^5/128
   signal reg_x3: std_logic_vector (16 downto 0) := "00000000000000000";
   signal reg_x5: std_logic_vector (16 downto 0) := "00000000000000000";

--Señal para registrar el valor de x - x^3/6 + x^5/120
   signal reg_suma: std_logic_vector (15 downto 0) := "0000000000000000";
begin
…
…
end Behavioral;
```

Figura 5.29. Arquitectura en VHDL y señales creadas. Diseño mediante la serie de Taylor.

Una vez están definidas las señales, se pasa a los procesos. El primer proceso se basa en relacionar el ángulo con otro del primer cuadrante y pasarlo a radianes.

```
process(clk,reset)
   variable bus_dir, dir: integer;
   variable x: std_logic_vector(11 downto 0):= "000000000000";
begin
   if reset = '1' then
      reg_xa <= "000000000000";
   elsif clk'event and clk = '1' then
      bus_dir := conv_integer(angulo);
--Miramos en que cuadrante estamos para sacar la relación correspondiente
--con un ángulo del primer cuadrante
      if bus_dir > 90 and bus_dir < 181 then         --II cuadrante
         dir := 180 - bus_dir;
      elsif bus_dir > 180 and bus_dir < 271 then     --III cuadrante
         dir := bus_dir - 180;
      elsif bus_dir > 270 and bus_dir < 361 then     --IV cuadrante
         dir := 360 - bus_dir;
      elsif bus_dir > 360 then
         dir := bus_dir - 360;
      else                                           --I cuadrante
         dir := bus_dir;
      end if;
      reg_bus_dir_a <= bus_dir;     --ángulo verdadero (0-360)
      reg_bus_dir_Ia <= dir;        --ángulo correspondiente al I cuadrante
      x := angulos_radianes(dir);   --ángulo en radianes del I cuadrante
      reg_xa <= x;                  --registro del ángulo en radianes
   end if;
end process;
```

Figura 5.30. Estudio de cuadrantes y primer registro. Diseño mediante la serie de Taylor.

Los dos siguientes procesos simplemente sirven para registrar las señales pertinentes y así poder usarlas cuando sea necesario.

```
process(clk,reset)
begin
   if reset = '1' then
      reg_xb <= "000000000000";
   elsif clk'event and clk = '1' then
      reg_xb <= reg_xa;                 --Retraso 1 clock ang(rad)
      reg_bus_dir_Ib <= reg_bus_dir_Ia; --Retraso 1 clock ang(I cuadrante)
      reg_bus_dir_b <= reg_bus_dir_a;   --Retraso 1 clock ang(verdadero)
   end if;
end process;

process(clk,reset)
begin
   if reset = '1' then
      reg_xc <= "000000000000";
   elsif clk'event and clk = '1' then
      reg_xc <= reg_xb;                 --Retraso 2 clock ang(rad)
      reg_bus_dir_Ic <= reg_bus_dir_Ib; --Retraso 2 clock ang(I cuadrante)
      reg_bus_dir_c <= reg_bus_dir_b;   --Retraso 2 clock ang(verdadero)
   end if;
end process;
```

Figura 5.31. Segundo y tercer registro para retrasar señales. Diseño mediante la serie de Taylor.

El siguiente proceso sirve para empezar a operar. La primera operación consta en conseguir x^2 y x^3. Aunque x^2 no se necesite como término para la serie, es necesario hallarlo para conseguir x^3. Una vez hallado se conserva porque sirve para luego hacer x^5.

```
process(clk,reset)
   variable x2: std_logic_vector (23 downto 0):= "000000000000000000 … 0";
   variable x3: std_logic_vector (35 downto 0):= "000000000000000000 … 0";
begin
   if reset = '1' then
      reg_x2_int <= "000000000000000000000000";
      reg_x3_int <= "000000000000000000000000000000000000";
   elsif clk'event and clk = '1' then
      x2 := reg_xa * reg_xa;
      x3 := reg_xa * reg_xa * reg_xa;
      reg_x2_int <= x2;
      reg_x3_int <= x3;
   end if;
end process;
```

Figura 5.32. Cálculo de x^2 y x^3. Diseño mediante la serie de Taylor.

El siguiente paso es calcular el valor de x^5. Posteriormente hay que hacer la división a los términos de x^3 y x^5. Esta división es entre 6 y 120 respectivamente, pero al no ser estos números potencias de dos, no podemos hacerla, por tanto, se divide entre 8 y 128. El error cometido se resolverá más adelante gracias a una función que ha sido creada.

```
process(clk,reset)
   variable x3: std_logic_vector (35 downto 0):= "000000000000000000 … 0";
   variable x5: std_logic_vector (59 downto 0):= "000000000000000000 … 0";
   variable x3_aprox: std_logic_vector (16 downto 0):= "00000000000000000";
   variable x5_aprox: std_logic_vector (16 downto 0):= "00000000000000000";
   variable x3_div: std_logic_vector (16 downto 0):= "00000000000000000";
   variable x5_div: std_logic_vector (16 downto 0):= "00000000000000000";
begin
   if reset = '1' then
      reg_x3 <= "00000000000000000";
      reg_x5 <= "00000000000000000";
   elsif clk'event and clk = '1' then
      x3 := reg_x3_int;                  --x^3
      x5 := reg_x3_int * reg_x2_int;     --cálculo de x^5

--Al hacer las multiplicaciones el vector se queda justificado a la
--derecha. Como ahora voy a dividir, que es lo mismo que meter ceros por la
--izquierda, quiero tener mi vector justificado a la izquierda primero para
--poder hacer bien la división. La función aproximar se encargará de
--justificar el vector a la izquierda.
--Esto se hace realmente por el formato de almacenamiento de datos que
--estamos usando. Para saber realmente donde está la coma de los decimales
      x3_aprox := aproximar ("00000000000000000000000000" & x3);
      x5_aprox := aproximar (x5);

--Divido entre 8 y 128 respectivamente
      x3_div := "000" & x3_aprox(16 downto 3);
      x5_div := "0000000" & x5_aprox(16 downto 7);

--Registro los valores de las divisiones
      reg_x3 <= x3_div;
      reg_x5 <= x5_div;
   end if;
end process;
```

Figura 5.33. Divisiones de x^3 y x^5. Diseño mediante la serie de Taylor.

Una vez hechas las divisiones, ya están disponibles todos los términos necesarios de la serie de Taylor para poder sumar. Como se vio en el apartado 5.5.1, al multiplicar la coma se descoloca. Por tanto, antes de empezar a sumar hay que recolocar la coma. Para que así las comas de los distintos términos queden alineadas, y cada posición del vector de un término corresponda con la posición del vector de otro término.

Aquí se ve porque había que justificar los vectores a la izquierda a la hora de dividir, si no se hubiera hecho esto, ahora no se podría localizar la posición que esta ocupando la coma.

La recolocación de los distintos términos, la suma de estos y además, la corrección del error cometido al dividir se harán bajo la función *suma*. A la cual se dedica un apartado entero para explicarla detalladamente. Ahora simplemente se muestra el último proceso que se encarga de sacar el valor final del seno.

```
 process(clk,reset)
   variable suma1: std_logic_vector (15 downto 0):= "0000000000000000";
   variable flag: std_logic := '0';
 begin
   if reset = '1' then
      reg_suma <= "0000000000000000";
   elsif clk'event and clk = '1' then
--Introducimos en la funcion todas las variables que serán necesarias
      suma1 := suma(reg_bus_dir_Ic,reg_xc,reg_x3,reg_x5);
--Hacer CA2 para los angulos (181-360)
--Sirve para sacar los angulos directamente con su signo correspondiente,
--independientemente de si es positivo o negativo
      if reg_bus_dir_c > 180 and reg_bus_dir_c < 361 then
         suma1 := suma1 - '1';
         for i in 0 to 15 loop
            if suma1(i) = '1' and flag = '0' then
               flag := '1';
            elsif flag = '1' then
               suma1(i) := not suma1(i);
            end if;
         end loop;
      end if;
--Registramos el valor obtenido
      reg_suma <= suma1;
   end if;
 end process;
--Sacamos el seno por el puerto de salida
 bus_datos <= reg_suma;
end Behavioral;
```

Figura 5.34. Suma de términos y resultado. Diseño mediante la serie de Taylor.

Este último proceso se usa para realizar la suma de todos los términos correspondientes para obtener el resultado.

El resultado debe de salir con signo positivo o negativo, para ello, se tiene que estudiar en qué cuadrante se está. Esto simplemente se hace con una condición de si el ángulo pertenece al intervalo (180,360], y si es así se hace el ca2 del resultado, poniendo así el valor en negativo. A la hora de hacer el ca2, se aprecia que al resultado obtenido se le ha restado uno, esto simplemente se ha hecho porque en ca2, la parte negativa del rango tiene un digito más que la positiva. Por ejemplo, teniendo 8 bits sería de -128 a 127, y en realidad se quiere ir de -127 a 127, para así tener exactamente los mismos resultados independientemente del signo del seno.

5.5.4. Función *aproximar*

Esta función sirve para justificar un vector a la izquierda del todo, es decir, se encargará de coger el primer bit cuyo valor sea ‘1’ y lo coloca en la posición más significativa del nuevo vector, lo siguiente es copiar el resto del vector. Para ello lo que se hace es recorrer el vector de izquierda a derecha, y cuando se encuentra el primer ‘1’, se copia el vector desde ahí hasta el final y se mete en una variable auxiliar.

```
function aproximar (valor: std_logic_vector(59 downto 0)) return
std_logic_vector is
   variable cont : integer;
   variable flag : std_logic;
   variable x    : std_logic_vector(59 downto 0);
   variable aux  : std_logic_vector(59 downto 0);
begin
   x    := "000000000000000000000000000000000000000000000000000000000000";
   aux  := "000000000000000000000000000000000000000000000000000000000000";
   cont := 59;
   flag := '0';
   x    := valor;
   for i in 59 downto 0 loop            --Recorremos el vector
      if x(i)='1' and flag='0' then     --Si encontramos el primer '1'
         flag     := '1';               --Aviso de que ha sido encontado
         aux(cont):= x(i);              --Copiamos en variable auxiliar
         x(i)     := '0';
         cont     := cont - 1;
      elsif flag='1' then               --Para el resto del vector
         if x(i) /= '1' and x(i) /= '0' then
            x(i) := '0';    --Por si acaso hay alguna posición del
         end if;            --vector indefinida la ponemos a cero
         aux(cont):= x(i);              --Seguimos copiando en variable aux.
         x(i)     := '0';
         cont     := cont - 1;
      end if;
   end loop;
   return aux(59 downto 43);            --Devolvemos el resultado (17 bits)
end;
```

Figura 5.35. Función aproximar.

Debido a que se trabaja con longitudes de 16 bits, no hace falta utilizar toda la longitud del vector auxiliar. Con 16 bits bastaría, aun así, se coge un bit más para aumentar la precisión.

Para ver visualmente una demostración de lo que se está llevando a cabo con esta función se ha realizado un pequeño esquema de ello, donde la primera columna es la variable *x* y la segunda la variable *aux*.

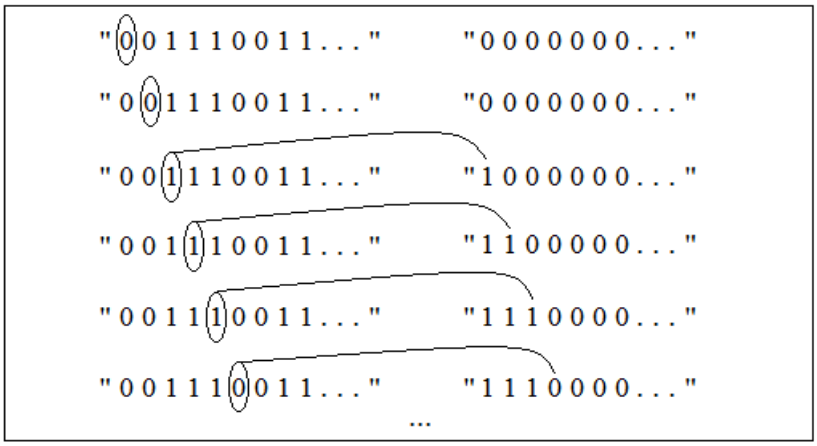


Figura 5.36. Visualización función aproximar.

5.5.5. Función *suma*

Esta función va a tener tres cometidos diferentes. El primero es recolocar los términos de entrada, después corregir el error cometido a la hora de dividir y finalmente, sumar todo para obtener el resultado final.

Tanto recolocar los términos como corregir el error depende del ángulo introducido. Por tanto, se necesita un CASE con noventa posibilidades, es decir, los 90 grados del primer cuadrante.

Recolocación de términos

Esta recolocación no es complicada, para poder ver como se realiza se usa un ejemplo de cómo habría que recolocar los términos para un ángulo concreto, por ejemplo, para 30º. En primer lugar se recuerda el formato de almacenamiento decimal utilizado, para tener en mente cuales son las posiciones existentes.

STD_LOGIC_VECTOR (15 downto 0)																
Posición	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
Valor	2^3	2^2	2^1	2^0	2^-1	2^-2	2^-3	2^-4	2^-5	2^-6	2^-7	2^-8	2^-9	2^-10	2^-11	2^-12
	8	4	2	1	0,5	0,25	0,125	0,0625	0,0313	0,0156	0,0078	0,0039	0,00195	0,00098	0,0004883	0,0002441
Vector	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Tabla 5.8. Formato de almacenamiento decimal.

Si se coge el valor obtenido de $x^3/8$ (30º) se tiene un vector de bits concreto con información almacenada en él. Se calcula su valor real manualmente, el cual tiene que ser 0.0179. Visualizando la tabla anterior, el valor inmediatamente inferior a 0.0179 es 0.0156, correspondiente a la posición -6. Por tanto, el vector se tiene que recolocar de tal manera que su primer digito que sea un '1' caiga en la posición -6. Sabiendo que los tres primeros valores del vector son '0' por la división entre 8 realizada, lo único que hay que hacer es introducir ceros desde la posición 3 a la -2. Quedando ceros de 3 a -5 y de -6 a -12 el vector correspondiente.

Puede ser que esto parezca una labor demasiado tediosa, pero manejando una simple hoja de Excel se resuelve bastante rápido. Debido en gran medida, a que se pueden ver todos los resultados en una tabla, simplemente es ir calculando escalonadamente la posición que debe ocupar cada vector. A continuación. Se muestran las tablas de los dos términos, donde se ve la posición que debe ocupar el vector según el ángulo en el que estemos.

Recolocación de $x^3/8$:

Angulo(º)	0-7	8-9	10-11	12-14	15-18	19-22	23-28	29-36	37-45	46-57	58-72	73-90
Posición	-13	-12	-11	-10	-9	-8	-7	-6	-5	-3	-2	-1

Recolocación de $x^5/128$:

Angulo(º)	0-28	29-32	33-37	38-43	44-49	50-57	58-65	66-75	76-86	87-90
Posición	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4

Corrección del error cometido en las divisiones

También hay que corregir el error que se ha cometido a la hora de dividir por los factoriales, ya que si no, el resultado no es exacto. Debido a que hay que crear un CASE para todo el rango posible de resultados, se aprovecha para corregir el error según el ángulo.

Para ello se ha utilizado una nueva hoja de Excel, en la cual se ha calculado la diferencia entre hacer la división mediante el método descrito en el diseño o hacerla directamente, y esa diferencia es la que se ha tenido en cuenta para corregir el error cometido.

Grados	x(rad)	x3/3!	x3/8	x3/3! - x3/8	x5/5!	x5/128	x5/5! - x5/128	Seno
1	0,0175	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0173
2	0,0349	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0349
3	0,0524	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0522
…	…	…	…	…	…	…	…	…
29	0,5061	0,0216	0,0162	0,0054	0,0003	0,0003	0,0000	0,4849
30	0,5236	0,0239	0,0179	0,0060	0,0003	0,0003	0,0000	0,5000
31	0,5411	0,0264	0,0198	0,0066	0,0004	0,0004	0,0000	0,5156
32	0,5585	0,0290	0,0218	0,0073	0,0005	0,0004	0,0000	0,5300
…	…	…	…	…	…	…	…	…
85	1,4835	0,5442	0,4081	0,1360	0,0599	0,0561	0,0037	0,9963
86	1,5010	0,5636	0,4227	0,1409	0,0635	0,0595	0,0040	0,9976
87	1,5184	0,5835	0,4376	0,1459	0,0673	0,0631	0,0042	0,9985
88	1,5359	0,6038	0,4529	0,1510	0,0712	0,0668	0,0045	0,9990
89	1,5533	0,6247	0,4685	0,1562	0,0754	0,0707	0,0047	0,9995
90	1,5708	0,6460	0,4845	0,1615	0,0797	0,0747	0,0050	1,0000

Tabla 5.9. Cálculo de errores en las divisiones de los términos.

El resultado del seno entonces queda de aplicar la siguiente fórmula:

$$seno(x) = x - \left[\frac{x^3}{8} + \left(\frac{x^3}{3!} - \frac{x^3}{8}\right)\right] + \left[\frac{x^5}{128} + \left(\frac{x^5}{5!} - \frac{x^5}{128}\right)\right]$$

Si se simplifica, vemos que es equivalente a la serie de Taylor:

$$seno(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!}$$

El valor de las restas de $\left(\frac{x^3}{3!} - \frac{x^3}{8}\right)$ y $\left(\frac{x^5}{5!} - \frac{x^5}{128}\right)$ ha sido pasado al formato correspondiente en binario para poderlo sumar. También para este proceso, se ha seguido trabajando con Excel, ya que gracias a una macro diseñada el proceso se vuelve prácticamente inmediato. Una vez están los resultados en forma de vector simplemente hay que meterlos al CASE para la suma.

Sumar los términos

Finalmente, el último paso que queda en la función es sumar todos los términos correspondientes, y devolver el valor del seno. Debido a la longitud usada en los vectores no se ha introducido todo el código de la función, ya que quedaría muy engorroso y no se entendería bien. Por ello, simplemente se ha expuesto la estructura de la función y se ha añadido el ejemplo para el ángulo de 30º.

```
function suma (angulo: integer; x: std_logic_vector(11 downto 0);x3:
std_logic_vector(16 downto 0); x5: std_logic_vector(16 downto 0)) return
std_logic_vector is
   variable suma : std_logic_vector (15 downto 0):= "0000000000000000";
begin
   case angulo is
      when (0) => … ;
      when (1) => … ;
      …
      when (30)=>    suma := (x & "0000") - ((("000000") & x3(16 downto 7))
                     + "0000011000") + ((("00000000") & x5(16 downto 9))
                     +"00000");
      …
      when (90) => … ;
      when others => suma := "0000000000000000";
   end case;
   return suma;
end;
```

Figura 5.37. Función suma.

Se ve como el primer término de la suma es la *x*, a la cual se le ponen 4 ceros por la derecha, ya que nuestro ángulo de entrada en radianes tenía 12 bits, sin embargo, ahora el resultado tiene 16 bits.

Luego se resta el término correspondiente a x^3, donde primero se le colocan 6 ceros en la izquierda para llevarlo a la posición correcta. Luego se le suma el vector "11000" cuyo valor es 0.006 tal y como se ve en la *tabla 5.9 y* finalmente se coloca también el término para x^5, y se le suma 0 porque el error cometido es inferior a las milésimas.

Una vez calculado el valor de la suma, la función lo retorna al diseño, el cual lo almacena en un puerto de salida.

Capítulo 6

Simulación, validación e implementación

En este capítulo se describe la validación realizada del diseño. Para ello se realizan varios tipos de pruebas. El primer paso es realizar las simulaciones necesarias para corroborar el funcionamiento de los diseños, posteriormente se hace un estudio de errores para saber qué porcentaje de error se ha cometido en cada diseño. Se lleva a cabo la síntesis para su implementación en FPGA y se hace un análisis de la velocidad de operación de cada diseño.

6.1. Herramientas utilizadas

En primer lugar se presentan las herramientas a partir de las cuales se va a trabajar para poder extraer las conclusiones y resultados necesarios. Se usa de nuevo la herramienta Xilinx ISE para la implementación del diseño, y también nuevos programas para estudiar los resultados obtenidos y hacer un estudio de errores.

6.1.1. Simulador digital ISim 0.61xd

Esta herramienta viene directamente integrada en el entorno de desarrollo Xilinx ISE, por tanto, en vez de instalar un simulador externo se ha trabajado con este. ISim es una herramienta de simulación y depuración para diseños en VHDL. Es un proceso asistido por computador, en el que se usa una metodología descendente compuesta por las siguientes fases [20]:

- Especificación: donde se establece la descripción de las funciones del circuito.
- Diseño arquitectural: en este paso se consigue una descripción sintetizable del circuito y de los bancos de prueba, donde se deben respetar las especificaciones necesarias. El resultado de esta fase es una síntesis de alto nivel y una simulación de carácter funcional.
- Diseño detallado: se utiliza una herramienta de síntesis automática, la cual genera una lista de puertas *(netlist)*, que viene proporcionada por el fabricante de los circuitos en los que se va a implementar el diseño. Modificaciones en el diseño pueden mejorar el rendimiento de este, obteniendo así mejores resultados en esta fase, ya se optimizando el tiempo o el área.
- Diseño físico: este proceso es el encargado de realizar el emplazamiento y rutado del dispositivo a programar. Son validados los retardos del circuito.
- Programación y pruebas: en esta fase se implementa el diseño arquitectural.

La herramienta ISim como simulador VHDL permite generar el comportamiento de los elementos internos y las salidas, tomando como entradas los estímulos correspondientes. Este comportamiento es estudiado mediante una validación funcional que consiste en comprobar si la respuesta del diseño coincide con las especificaciones de este. Además ofrece útiles funciones para el proceso de depuración, como son el uso de ficheros para interactuar con el exterior.

6.1.2. Software matemático MATLAB R2011a

Esta herramienta de software matemático ofrece un entorno de desarrollo integrado con un lenguaje de programación propio (lenguaje M). Entre sus prestaciones básicas tenemos: la manipulación de matrices, la representación de datos, el uso de algoritmos, la creación de interfaces de usuario y la comunicación con otros lenguajes y otros dispositivos hardware [21]. Además se pueden ampliar las funcionalidades de este programa mediante la adición de otros paquetes como: Simulink, Guide…

Particularmente, se ha utilizado esta herramienta para hacer un análisis de los errores cometidos, obteniendo así una representación gráfica de ellos, mediante la cual se ve en qué puntos los errores son mayores y en cuales menores.

6.2. Valores de entrada para la simulación

Las simulaciones se han realizado para los ángulos existentes en el siguiente intervalo $[0º - 360º] \in Z$. Estos valores se han utilizado en todos los diseños, lo único que cambia es el formato de almacenamiento. También se ha llevado a cabo la introducción de estímulos de diferentes maneras, tal y como se puede apreciar en los bancos de pruebas anexados. Se ha comprobado a introducir un ángulo de manera individual y obtener el resultado, a introducir una cadena de ángulos, y también, a introducir el intervalo entero de 0º a 360º, construyendo así una señal sinusoidal.

Además, se ha hecho un estudio del error absoluto cometido. Donde por normal general, se ha comparado los resultados obtenidos con un seno obtenido mediante la

herramienta matemática MATLAB. Este error se ha mostrado en %, y ha sido calculado con la siguiente fórmula:

$$Error\ (\%) = \left|\frac{V_{MATLAB} - V_{DISEÑO}}{V_{MATLAB}}\right| * 100$$

6.3. Simulaciones y cálculo de errores

En este apartado se estudian los resultados obtenidos para cada diseño, además del cálculo de errores para cada uno de ellos. Debido a la realización de tres diseños distintos, la explicación se divide en tres apartados. No se muestra el resultado de todos los ángulos, puesto que ocuparía demasiado, simplemente se van mostrando resultados de diferentes ángulos en cada diseño.

6.3.1. Cálculo del seno mediante aproximación por muestreo

El diseño por aproximación se ha realizado con tres longitudes de vector diferentes, por tanto, los resultados obtenidos son diferentes dependiendo de la longitud.

- **Longitud de 8 bits**

Tal y como se explicó en el capítulo 5, este diseño trabaja con 8 bits, por tanto, el resultado se expresa en enteros que van de -100 a 100, de tal manera que para que el resultado sea interpretado correctamente debe ser dividido entre 100. Además, recordar que el diseño constaba de distinto número de muestras (13, 25 y 37).

En la siguiente tabla se aprecian los resultados obtenidos en una serie de ángulos. En este primer caso, el estudio del error no se ha realizado respecto a un seno de matlab, ya que, el resultado que se hubiera obtenido se sabe que es un error bastante alto. Por tanto, el estudio del error se ha realizado respecto el diseño en el cuál se ha interpolado.

Áng.	Seno matlab (rad)	Resultados aproximación con 8 bits								
		13 muestras			25 muestras			37 muestras		
		Aprox.	Inter.	Error	Aprox.	Inter.	Error	Aprox.	Inter.	Error
1º	0,018	0,50	0,02	2400%	0,26	0,02	1200%	0,17	0,02	750%
37º	0,602	0,87	0,59	47%	0,71	0,60	18%	0,64	0,60	7%
66º	0,913	1,00	0,90	11%	0,97	0,91	7%	0,94	0,92	2%
81º	0,988	1,00	0,97	3%	1,00	0,99	1%	1,00	0,99	1%
90º	1,0000	1,00	1,00	0%	1,00	1,00	0%	1,00	1,00	0%
238º	-0,848	-0,87	-0,85	2%	-0,87	-0,85	2%	-0,87	-0,86	1%
307º	-0,798	-0,50	-0,78	36%	-0,71	-0,79	10%	-0,77	-0,79	3%
360º	0,000	0,00	0,00	0%	0,00	0,00	0%	0,00	0,00	0%

Tabla 6.1. Resultados del diseño por aproximación con 8 bits

Debido a que se está en el diseño de aproximación se dejan para más adelante los resultados obtenidos en la interpolación y se estudian solo los resultados aproximados. Se puede apreciar cómo según aumenta el número de muestras, el resultado está más cerca del valor real. Como es lógico, al haber mayor frecuencia de muestreo, la distancia entre las muestras es menor.

También se puede apreciar como para 90º y 360º, el resultado es exacto y por tanto, el error es del 0%. Esto se debe a que estos ángulos forman parte del número de muestras a partir de las cuales se hace el diseño.

En las siguientes gráficas se dibujan el seno aproximado, el seno interpolado y el error cometido al usar uno en vez del otro.

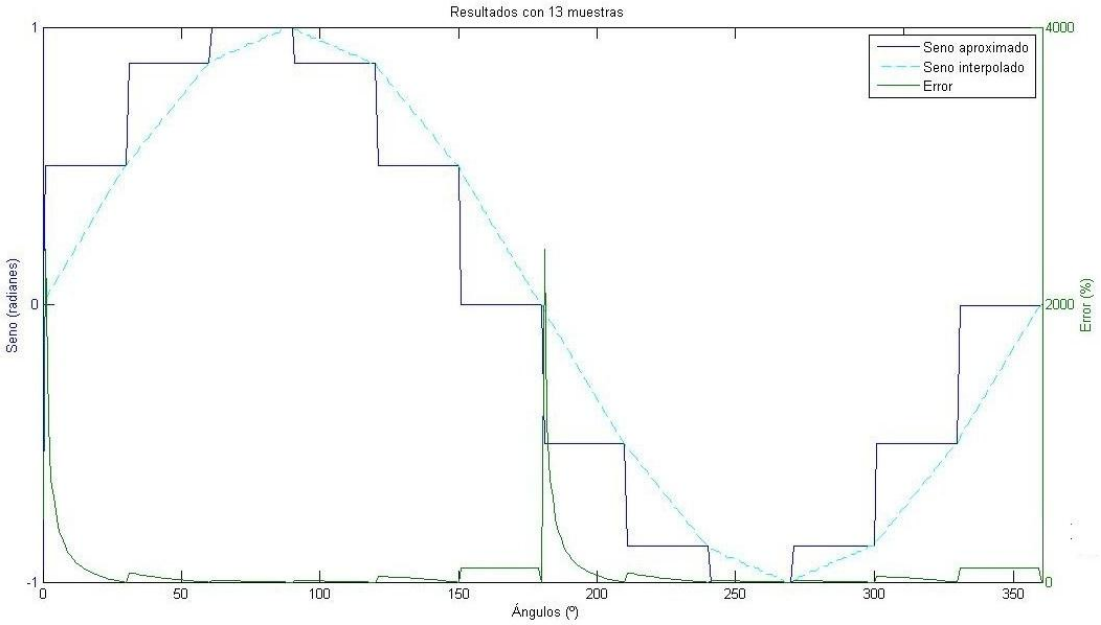


Figura 6.1. Resultados del diseño por aproximación con 13 muestras (8 bits)

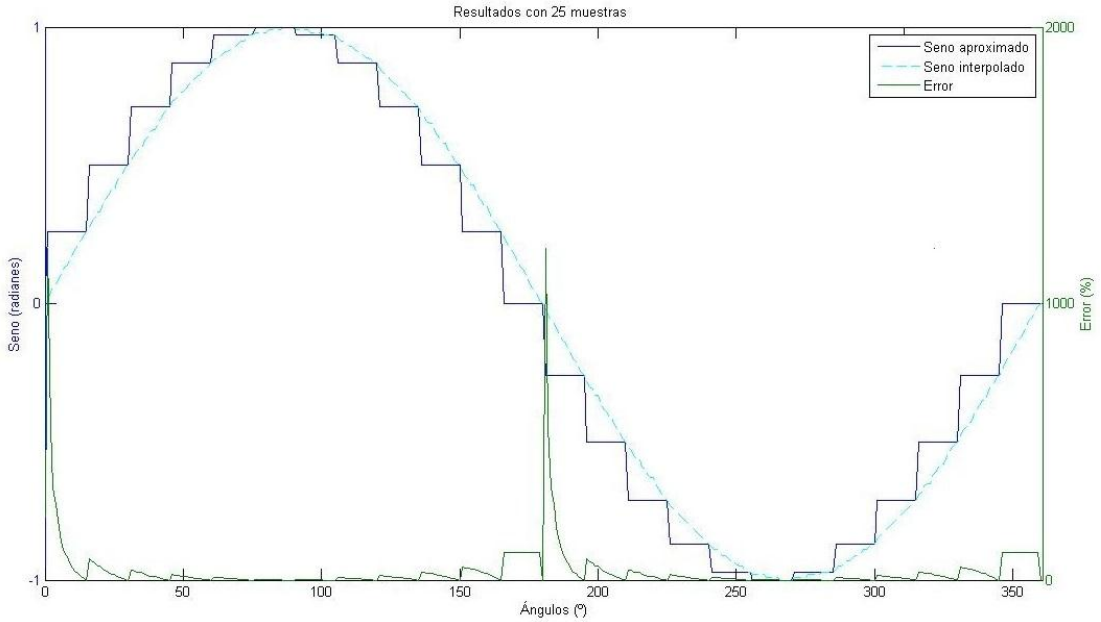


Figura 6.2. Resultados del diseño por aproximación con 25 muestras (8 bits)

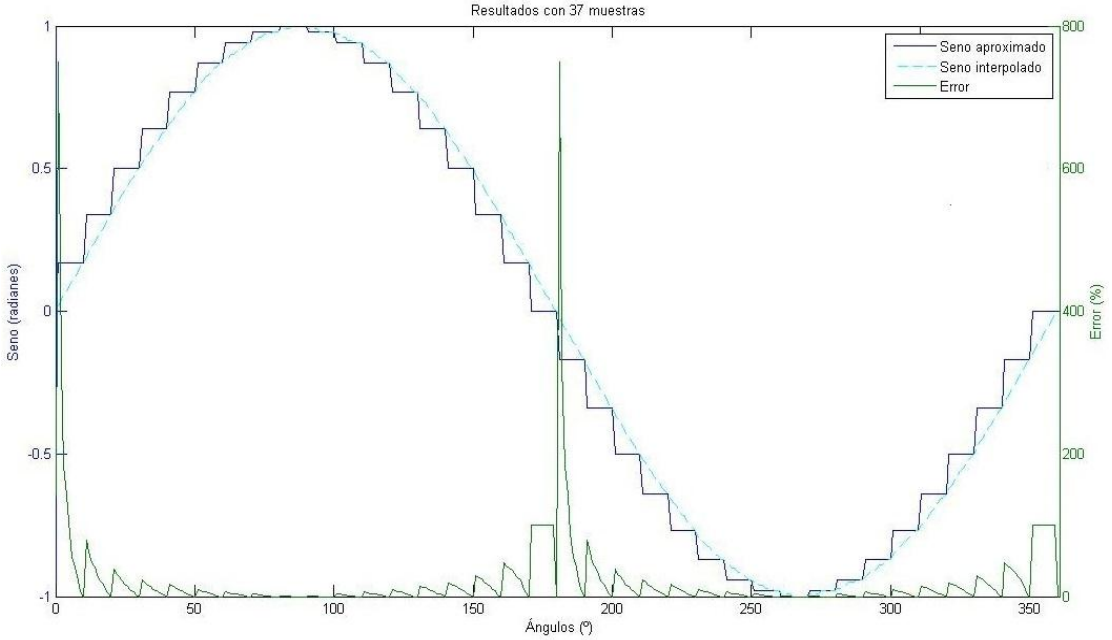


Figura 6.3. Resultados del diseño por aproximación con 37 muestras (8 bits)

Según aumenta el número de muestras se obtiene un seno más escalonado, lo que conlleva a que el error disminuya, aunque en ningún momento se obtiene un error aceptable. De hecho, la media ponderada de estos errores es 70.05%, 34.60% y 21.92%, para 13, 25 y 37 muestras respectivamente.

También se aprecia en las gráficas que los picos mayores de los errores se dan en torno a 0º, 180º y 360º, esto se debe a que el valor del seno en esos entornos es muy cercano a cero, por tanto, al dividir entre un valor que es prácticamente cero, hace que la división tienda hacia infinito, y consecuentemente el error aumente tanto.

- **Longitud de 16 bits**

Para esta otra longitud el resultado oscila entre -10.000 y 10.000. Por tanto, se divide entre 10.000 para interpretar el resultado. De igual modo se ha efectuado el diseño para distintos números de muestras. En la siguiente tabla, se aprecian los resultados obtenidos.

Áng.	Seno matlab (rad)	Resultados aproximación con 16 bits								
		13 muestras			25 muestras			37 muestras		
		Aprox.	Inter.	Error	Aprox.	Inter.	Error	Aprox.	Inter.	Error
10º	0,1736	0,5000	0,1667	200%	0,2588	0,1726	50%	0,1736	0,1563	11%
68º	0,9272	1,0000	0,9018	11%	0,9659	0,9193	5%	0,9397	0,9250	2%
128º	0,788	0,5000	0,7683	35%	0,7071	0,7812	9%	0,7660	0,7859	3%
177º	0,0523	0,0000	0,0499	100%	0,0000	0,0517	100%	0,0000	0,0520	100%
211º	-0,515	-0,8660	-0,5123	69%	-0,7071	-0,5139	38%	-0,6428	-0,5143	25%
322º	-0,6157	-0,5000	-0,5975	16%	-0,5000	-0,6104	18%	-0,5000	-0,6142	19%
360º	-2E-16	0,0000	0,0000	0%	0,0000	0,0000	0%	0,0000	0,0000	0%

Tabla 6.2. Resultados del diseño por aproximación con 16 bits

De nuevo se observa que por norma general según aumenta el número de muestras, los valores son más aproximados al valor correcto, y por tanto, disminuye el error cometido.

El conjunto de estos resultados quedan dibujados en las siguientes tres gráficas:

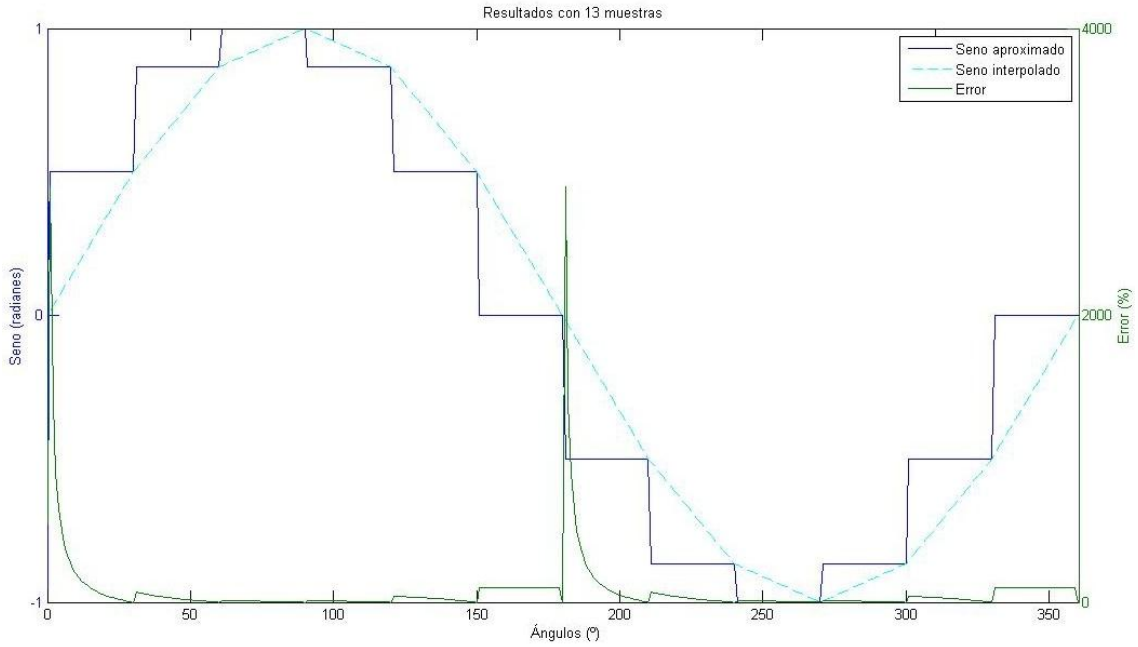


Figura 6.4. Resultados del diseño por aproximación con 13 muestras (16 bits)

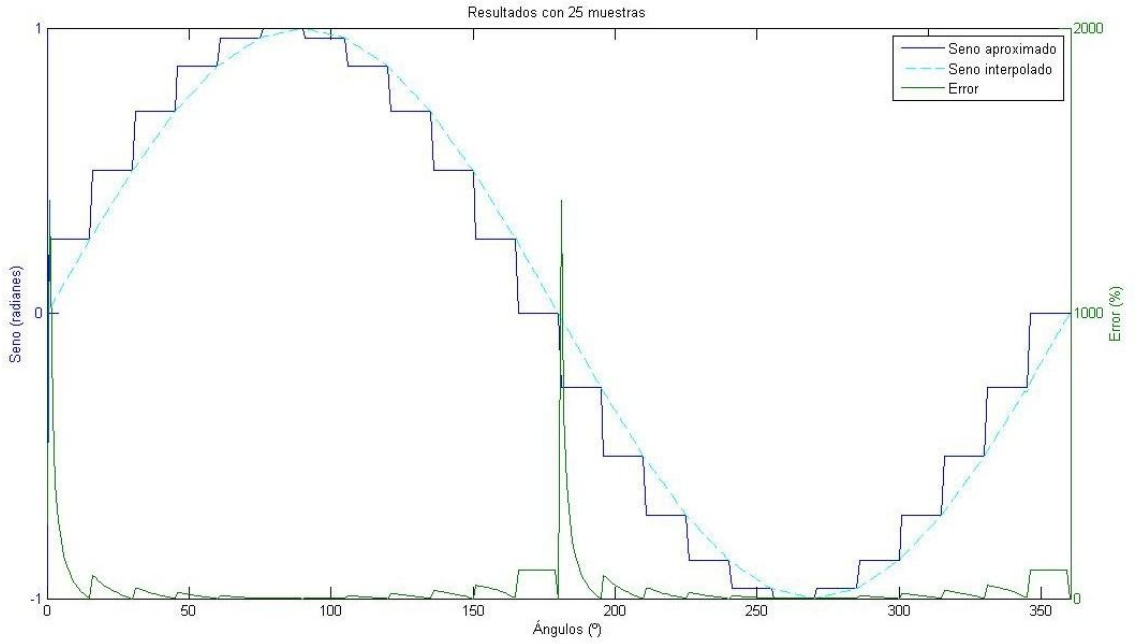


Figura 6.5. Resultados del diseño por aproximación con 25 muestras (16 bits)

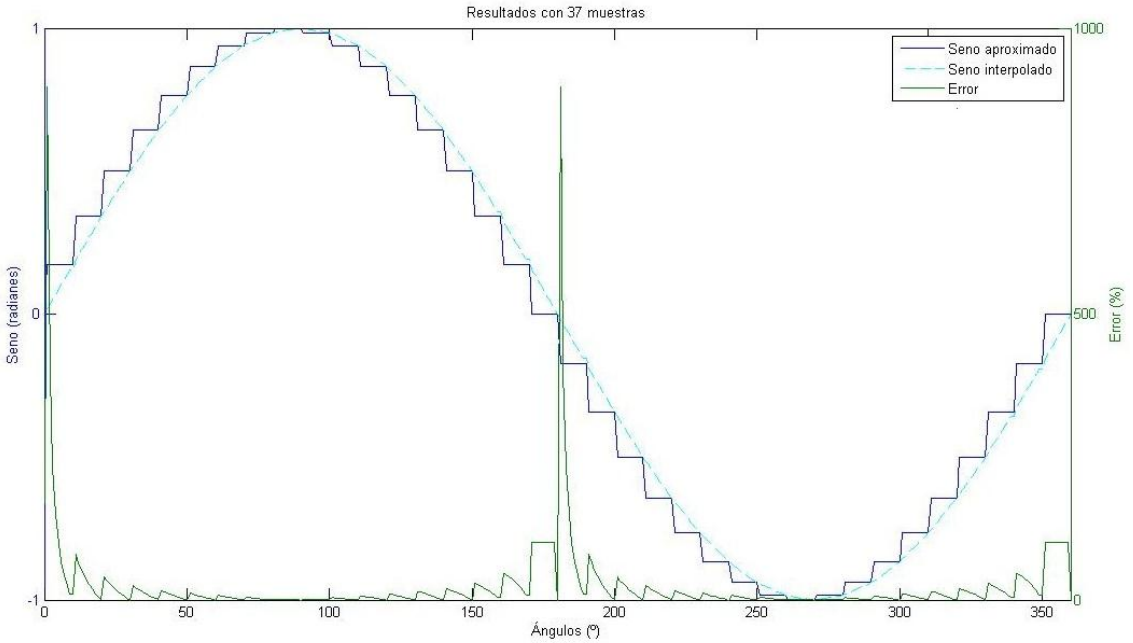


Figura 6.6. Resultados del diseño por aproximación con 37 muestras (16 bits)

Debido a la similitud con el anterior diseño, las gráficas obtenidas son semejantes, donde lo único que cambia son los valores de los errores, pero mínimamente. Este cambio tan pequeño en los errores se debe a que los valores obtenidos del seno van a tener un mayor número de decimales, pero no significa que los resultados vayan a ser más cercanos a un seno real. De hecho, sucede al contrario, al obtenerse mayor número de decimales, hay mayor diferencia entre lo obtenido y lo realmente exacto, por tanto, el error cometido crece ligeramente. El cálculo de la media ponderada de errores para los tres diseños respectivamente ha dejado los siguientes resultados: 76.88%, 37.46% y 24.47%.

- **Longitud de 32 bits**

Finalmente, la última longitud utilizada es de 32 bits. En este caso el resultado también serán números enteros, donde ahora irán desde -10^9 a 10^9. Dividiendo por tanto entre 10^9 para llevar a cabo una interpretación correcta. La siguiente tabla muestra algunos de los resultados:

Áng.	Seno matlab (rad)	Resultados aproximación con 32 bits								
		13 muestras			25 muestras			37 muestras		
		aprox.	Inter.	Error	aprox.	Inter.	Error	aprox.	Inter.	Error
23º	0,3907	0,500000	0,383333	30%	0,500000	0,387449	29%	0,500000	0,389414	28%
41º	0,6561	0,866025	0,634209	37%	0,707107	0,651878	8%	0,766044	0,655113	17%
54º	0,809	0,866025	0,792820	9%	0,866025	0,802458	8%	0,866025	0,806037	7%
60º	0,866	0,866025	0,866025	0%	0,866025	0,866025	0%	0,866025	0,866025	0%
75º	0,9659	1,000000	0,933013	7%	0,965926	0,965926	0%	0,984808	0,962250	2%
80º	0,9848	1,000000	0,955342	5%	1,000000	0,977284	2%	0,984808	0,984808	0%
85º	0,9962	1,000000	0,977671	2%	1,000000	0,988642	1%	1,000000	0,992404	1%
90º	1	1,000000	1,000000	0%	1,000000	1,000000	0%	1,000000	1,000000	0%

Tabla 6.3. Resultados del diseño por aproximación con 32 bits

En la tabla se puede apreciar el aumento de decimales usados, además de la tendencia similar de los resultados frente a los anteriores obtenidos. Las gráficas, de igual modo seguirán esta tendencia.

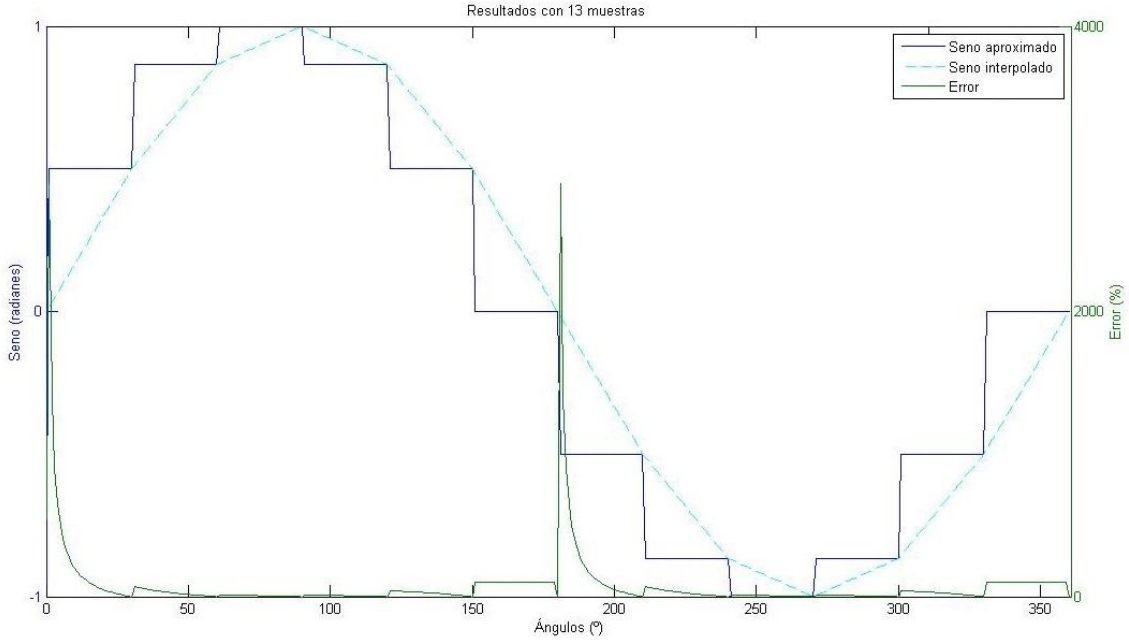


Figura 6.7. Resultados del diseño por aproximación con 13 muestras (32 bits)

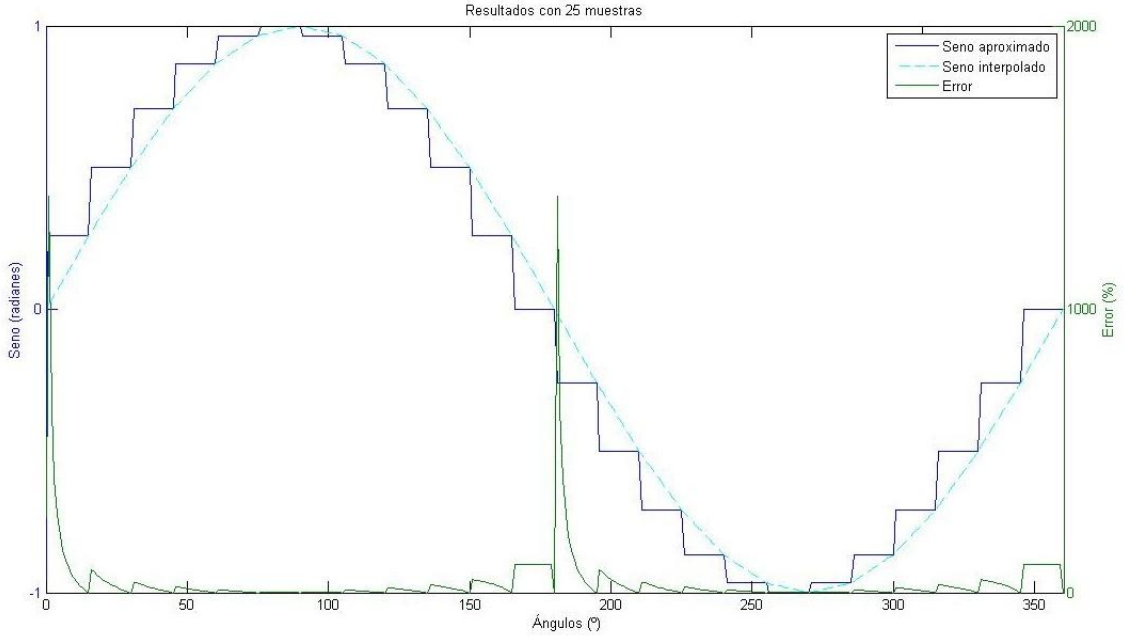


Figura 6.8. Resultados del diseño por aproximación con 25 muestras (32 bits)

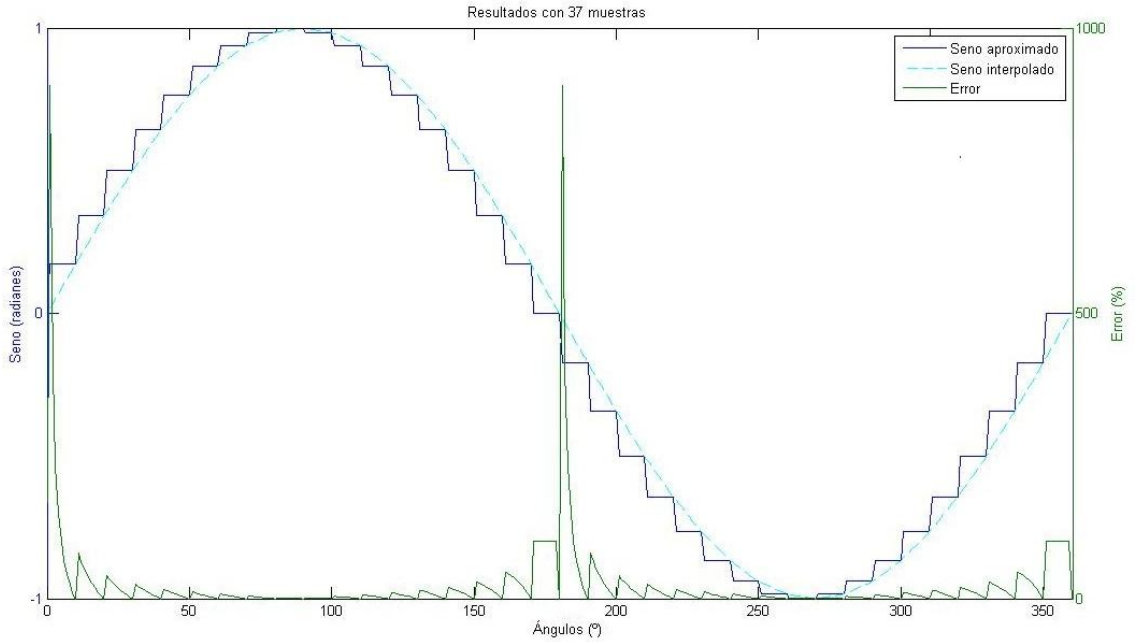


Figura 6.9. Resultados del diseño por aproximación con 37 muestras (32 bits)

En este último caso los valores medios para el cálculo de errores son de 76.97%, 37.56% y 24.57%. Se puede ver que son exactamente los mismos resultados que para 16 bits. Por tanto, esto significa que entre usar 16 bits ó 32 bits, no hay ninguna diferencia, puesto que los resultados son equivalentes. La coincidencia se debe a que a partir de cierto número de decimales los cambios son tan insignificantes que no se notan en el resultado.

Por tanto, la conclusión que se puede extraer es que ante dos diseños cuyos resultados son comunes, pero donde uno ocupa menos memoria, tiempo y área, además de ser sintetizable y el otro no, habrá que decantarse por el de 16 bits desechando el diseño de la aproximación por muestreo con 32 bits.

6.3.2. Cálculo del seno mediante interpolación lineal

De igual manera que para el diseño anterior, este diseño también es realizado para tres longitudes de vector distintas, obteniéndose así diferentes resultados. Este diseño si se ha comparado directamente con un seno de matlab, debido a que ahora el error cometido si va a estar dentro de un margen de error razonable.

- **Longitud de 8 bits**

Los resultados obtenidos y los errores cometidos para esta longitud se presentan en la siguiente tabla:

Áng.	Seno matlab (rad)	Resultados interpolación con 8 bits					
		13 muestras		25 muestras		37 muestras	
		Inter.(rad)	Error	Inter.(rad)	Error	Inter.(rad)	Error
4º	0,06976	0,07	0,3%	0,07	0,3%	0,07	0,3%
38º	0,61566	0,60	2,5%	0,62	0,7%	0,62	0,7%
77º	0,97437	0,95	2,5%	0,98	0,6%	0,97	0,4%
110º	0,93969	0,91	3,2%	0,93	1,0%	0,94	0,0%
147º	0,54464	0,53	2,7%	0,54	0,9%	0,54	0,9%
196º	-0,2756	-0,27	2,0%	-0,28	1,6%	-0,28	1,6%
254º	-0,9613	-0,94	2,2%	-0,97	0,9%	-0,96	0,1%
300º	-0,866	-0,87	0,5%	-0,87	0,5%	-0,87	0,5%
348º	-0,2079	-0,19	8,6%	-0,20	3,8%	-0,20	3,8%

Tabla 6.4. Resultados del diseño por interpolación con 8 bits

En las siguientes gráficas se pueden apreciar los resultados para todos los ángulos, además, se ve como el resultado de la gráfica interpolada frente a la del seno de matlab quedan prácticamente superpuestas, y esto se debe a la disminución del error cometido, como se ha podido ir apreciando en los resultados de la tabla.

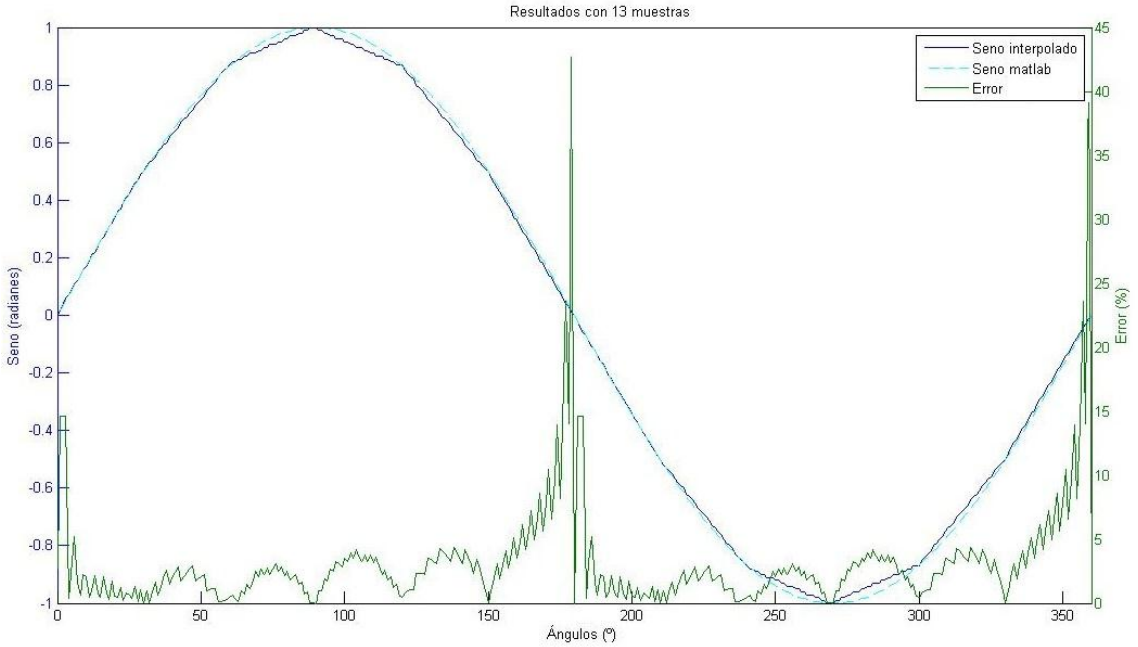


Figura 6.10. Resultados del diseño por interpolación con 13 muestras (8 bits)

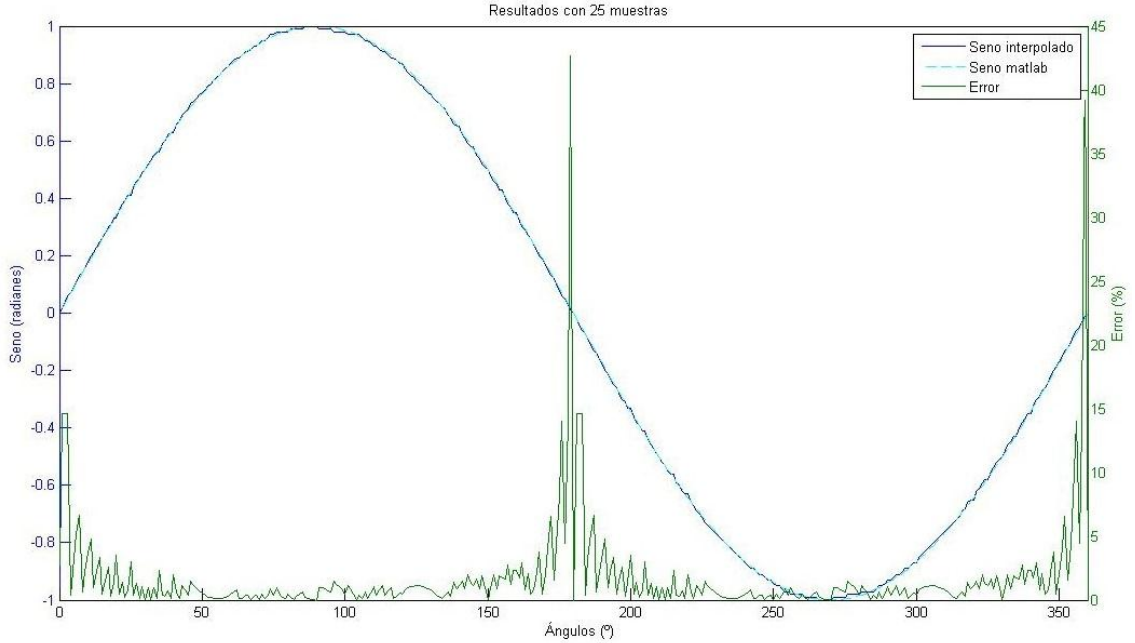


Figura 6.11. Resultados del diseño por interpolación con 25 muestras (8 bits)

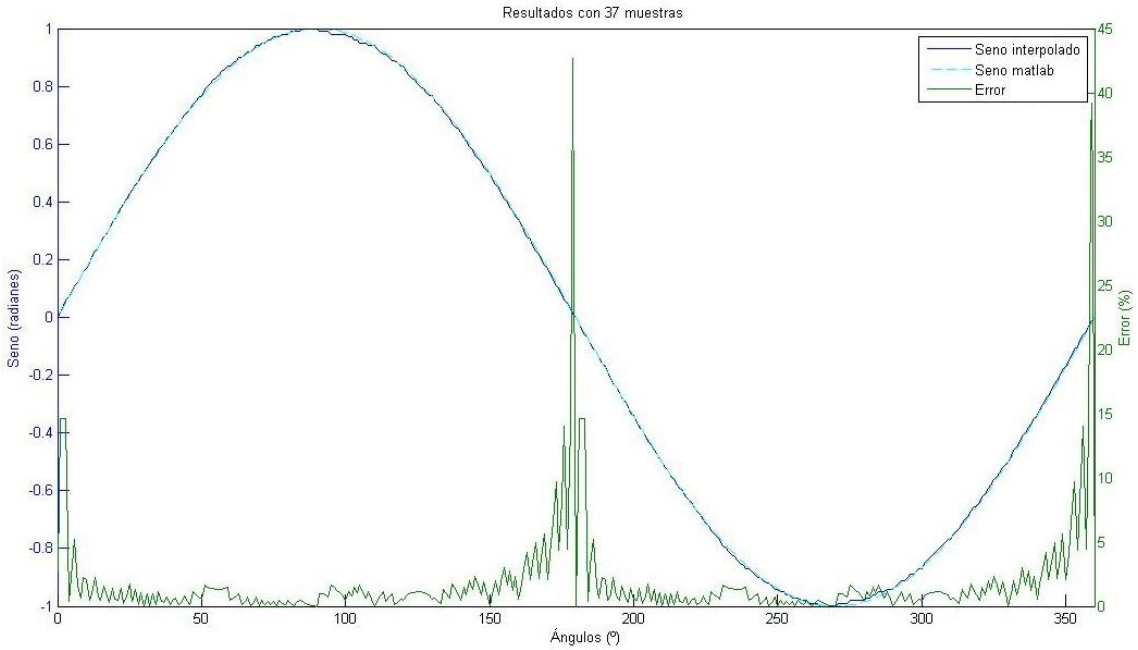


Figura 6.12. Resultados del diseño por interpolación con 37 muestras (8 bits)

En todas las gráficas se puede ver como el error ha disminuido considerablemente, obteniendose así un promedio de 3.13%, 1.71% y 1.81% respectivamente.

- **Longitud de 16 bits**

Repetimos el proceso nuevamente para esta longitud. Primero se muestra una pequeña tabla con algunos valores y posteriormente las gráficas obtenidas.

Áng.	Seno matlab (rad)	Resultados interpolación con 16 bits					
		13 muestras		25 muestras		37 muestras	
		Inter.(rad)	Error	Inter.(rad)	Error	Inter.(rad)	Error
6º	0,1045	0,1001	4,2%	0,1036	0,9%	0,1042	0,3%
78º	0,9781	0,9465	3,2%	0,9728	0,5%	0,9758	0,2%
94º	0,9976	0,9821	1,6%	0,9909	0,7%	0,9939	0,4%
109º	0,9455	0,9151	3,2%	0,9392	0,7%	0,9442	0,1%
154º	0,4384	0,4333	1,2%	0,4356	0,6%	0,4367	0,4%
173º	0,1219	0,1166	4,3%	0,1207	1,0%	0,1215	0,3%
246º	-0,9135	-0,8929	2,3%	-0,9060	0,8%	-0,9103	0,4%
318º	-0,6691	-0,6463	3,4%	-0,6656	0,5%	-0,6674	0,3%

Tabla 6.5. Resultados del diseño por interpolación con 16 bits

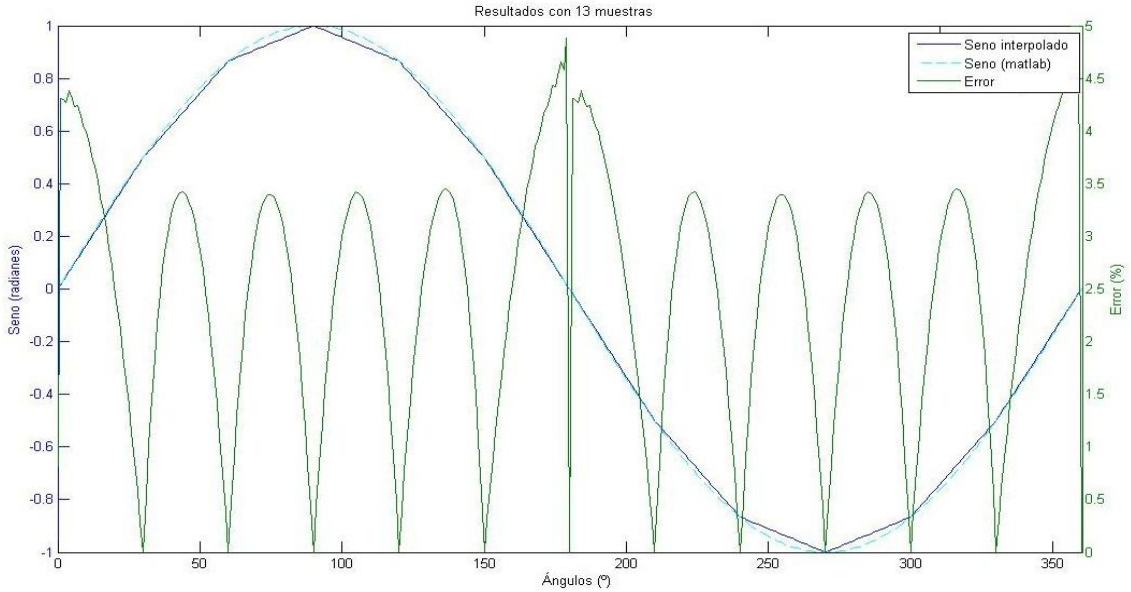


Figura 6.13. Resultados del diseño por interpolación con 13 muestras (16 bits)

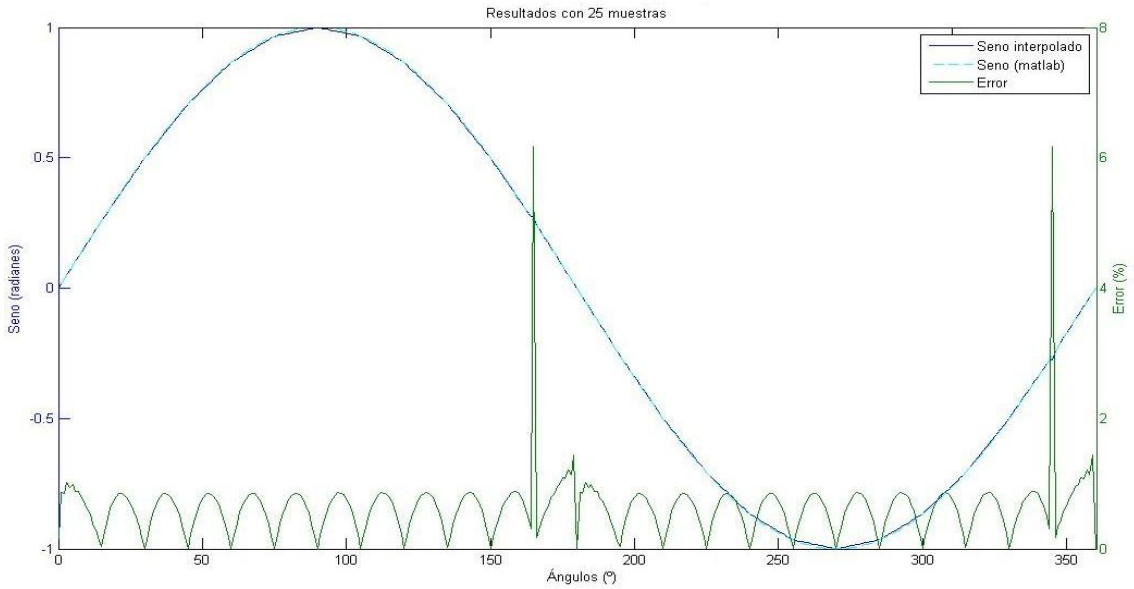


Figura 6.14. Resultados del diseño por interpolación con 25 muestras (16 bits)

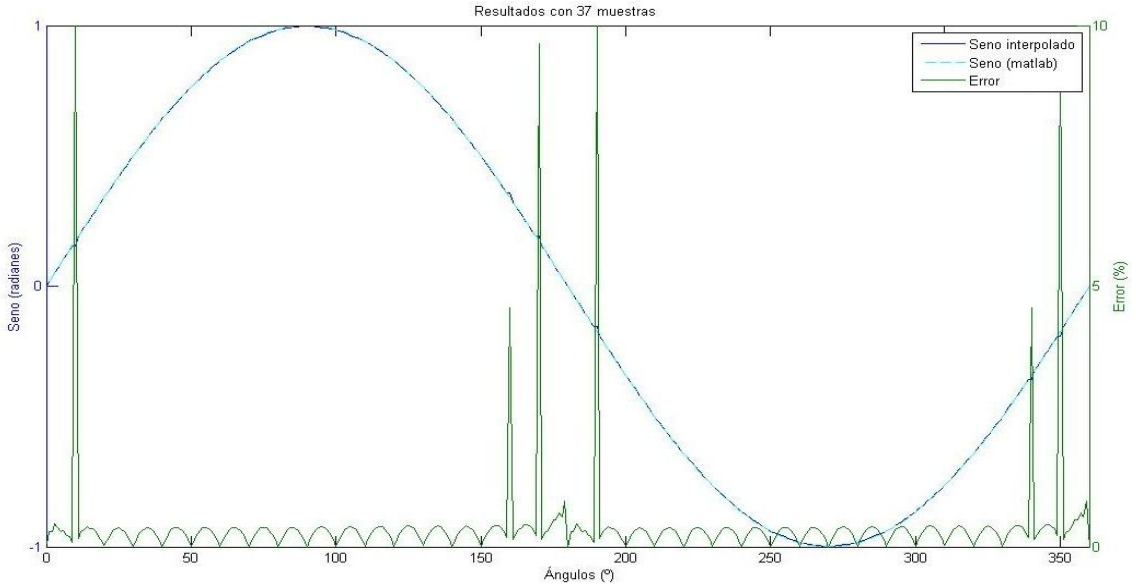


Figura 6.15. Resultados del diseño por interpolación con 37 muestras (16 bits)

Con esta longitud se ha obtenido un error medio de 2.51%, 0.63% y 0.40%, para 13, 25 y 37 muestras respectivamente.

- **Longitud de 32 bits**

Finalmente se estudia el error cometido para la longitud de 32 bits, a pesar de saber que este diseño no es sintetizable. Simplemente se hace para comparar los resultados respecto de las anteriores longitudes.

Áng.	Seno matlab (rad)	Resultados interpolación con 32 bits					
		13 muestras		25 muestras		37 muestras	
		Inter.(rad)	Error	Inter.(rad)	Error	Inter.(rad)	Error
11º	0,1908	0,183333346	3,9%	0,189800633	0,5%	0,190485379	0,2%
48º	0,7431	0,719615252	3,2%	0,738890516	0,6%	0,741393077	0,2%
137º	0,682	0,658610998	3,4%	0,679492534	0,4%	0,679764659	0,3%
198º	-0,309	-0,300000008	2,9%	-0,307055240	0,6%	-0,308345751	0,2%
225º	-0,7071	-0,683012714	3,4%	-0,707106781	0,0%	-0,704416028	0,4%
290º	-0,9397	-0,910683594	3,1%	-0,932625684	0,8%	-0,939692621	0,0%
346º	-0,2419	-0,233333324	3,6%	-0,241564442	0,1%	-0,240996962	0,4%

Tabla 6.6. Resultados del diseño por interpolación con 32 bits

Del mismo modo que para el diseño anterior mediante la aproximación por muestreo, ahora también se obtienen resultados semejantes entre usar una longitud de vector de 32 bits y de 16 bits. A continuación se puede corroborar con los resultados de las gráficas.

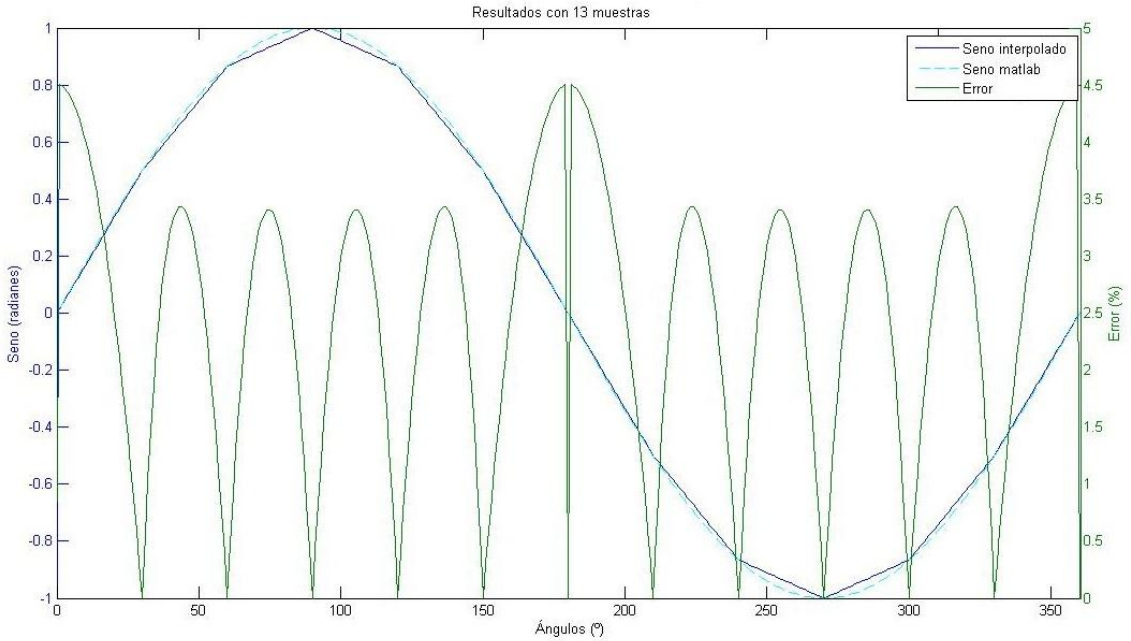


Figura 6.16. Resultados del diseño por interpolación con 13 muestras (32 bits)

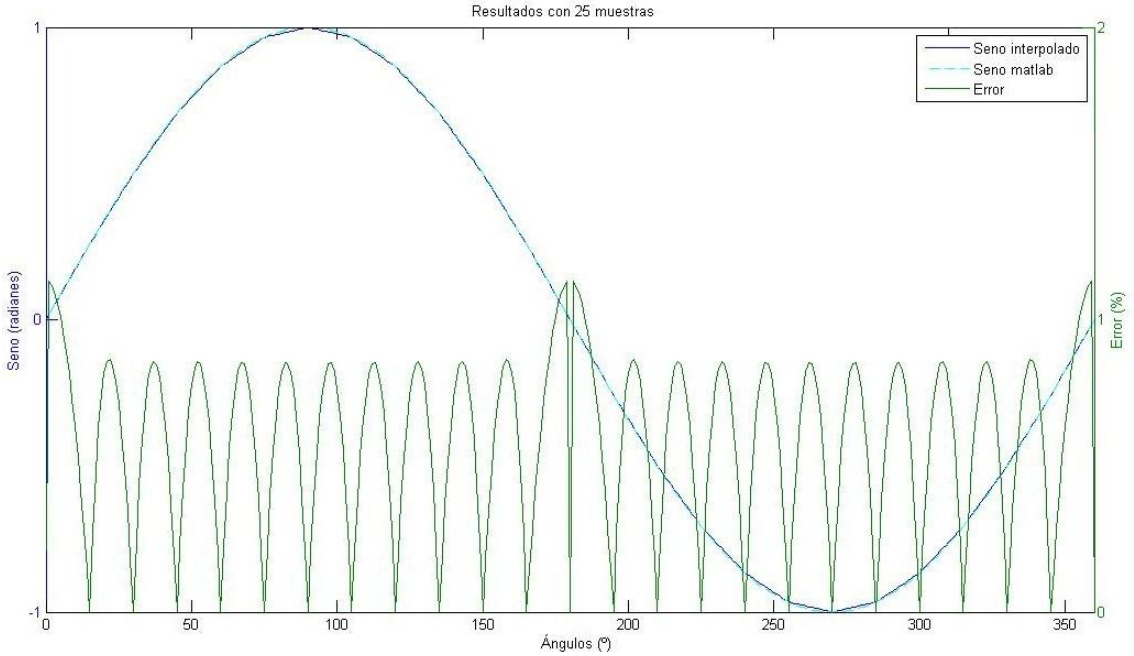


Figura 6.17. Resultados del diseño por interpolación con 25 muestras (32 bits)

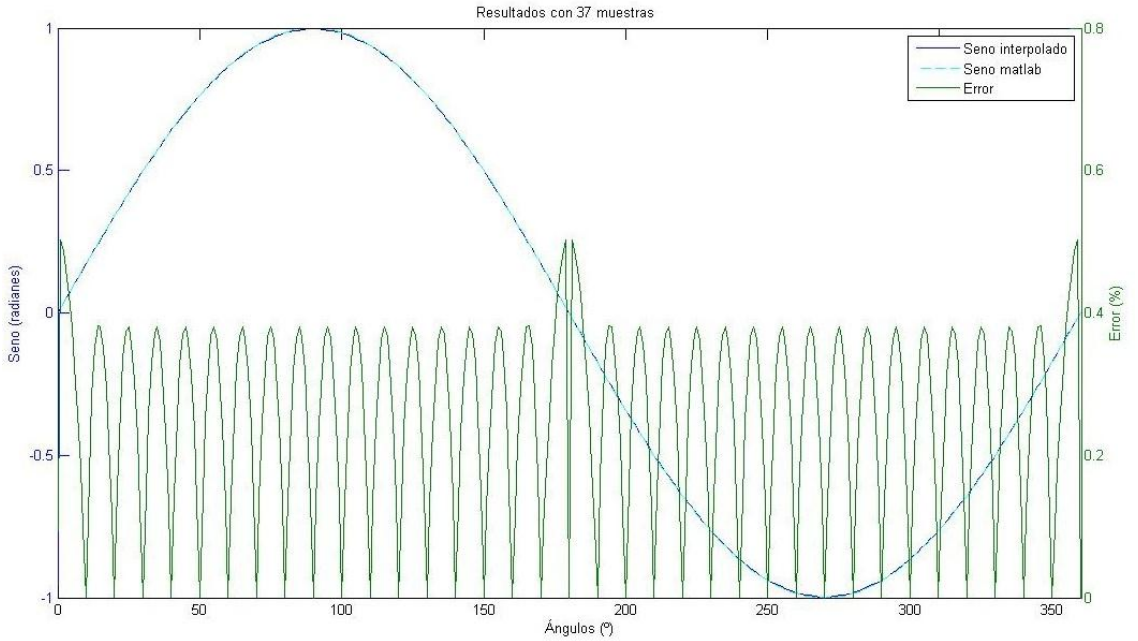


Figura 6.18. Resultados del diseño por interpolación con 37 muestras (32 bits)

Aunque los resultados obtenidos sean prácticamente iguales, se puede observar que el desarrollo visual del error en las gráficas es totalmente distinto. Para esta longitud se obtiene una regularidad prácticamente perfecta. Esto se debe al distinto diseño que se ha llevado a cabo para 32 bits.

El error promedio según el número de muestras utilizadas ha sido de 2.50%, 0.59% y 0.26% respectivamente.

6.3.3. Cálculo del seno mediante la serie de Taylor

El método mediante el cual se llega a la mejor optimización de los distintos diseños es la serie de Taylor. Aunque, desde un principio ha sido el que más complicaciones ha expuesto.

Áng.	Seno Matlab (rad)	Seno Taylor (rad)	Error
7º	0,1219	0,1218	0,04%
46º	0,7193	0,7192	0,01%
73º	0,9563	0,9563	0,00%
177º	0,0523	0,0522	0,17%
232º	-0,7880	-0,7881	0,01%
250º	-0,9397	-0,9397	0,00%
280º	-0,9848	-0,9846	0,02%
305º	-0,8192	-0,8191	0,01%
360º	0,0000	0,0000	0,00%

Tabla 6.7. Resultados del diseño mediante la serie de Taylor

En los ejemplos mostrados en la tabla se ve que los tres primeros decimales son siempre exactos, y el cuarto decimal si no lo es, se aproxima bastante a su valor correspondiente.

Para poder interpretar los resultados de este método se ha tenido que dividir por 2^n, donde *n* es el número de bits que hay después de la coma imaginaria en el vector. En este caso hay 16 bits de vector, de los cuales 4 bits son parte entera y 12 bits parte decimal, por tanto se divide por 2^{12} que es 4096. Obteniendo así los resultados correctos, que oscilan de -1.0 a 1.0.

Debido a que ahora solo se ha utilizado una longitud y que ya no es necesario un número de muestras a partir de las cuales se construye el diseño, solo se obtiene una gráfica.

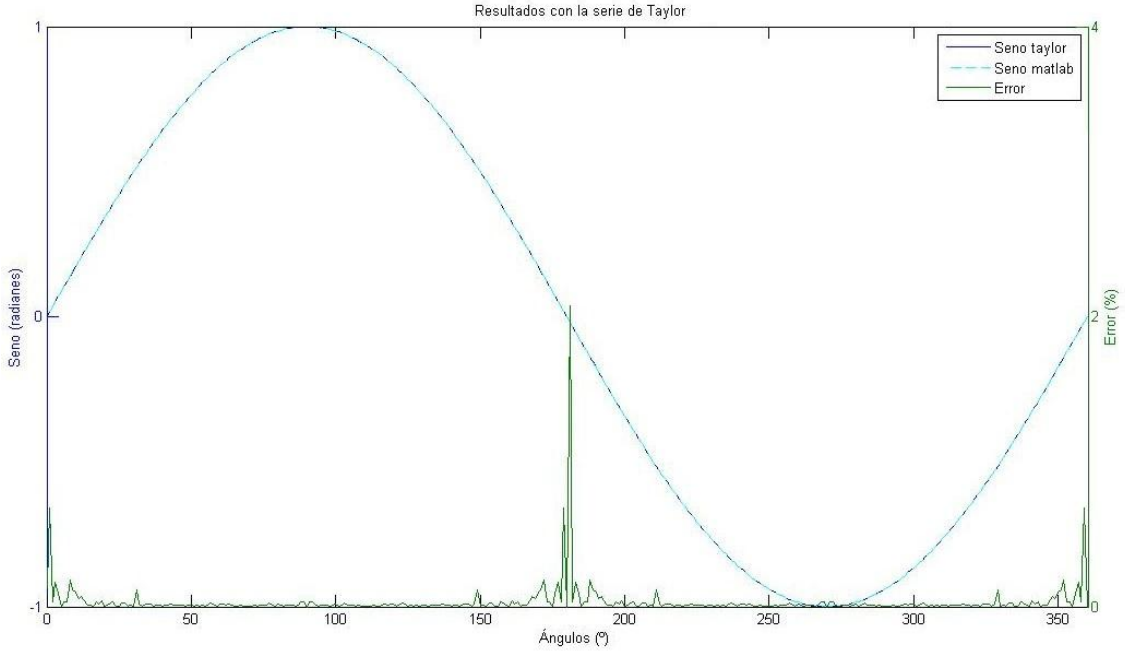


Figura 6.19. Resultados del diseño mediante la serie de Taylor

Se puede apreciar como la línea que marca el error esta todo el rato prácticamente en cero, excepto cuando se aproxima a los puntos críticos, que son cuando el seno se aproxima a cero. En este diseño se ha obtenido un error medio de 0.033%, logrando así resultados muchos más óptimos que en los diseños previos.

6.3.4. Resumen del cálculo de errores

En este apartado simplemente aparece un resumen de los resultados obtenidos en el cálculo de errores. Los valores mostrados se corresponden con el error medio cometido entre todos los ángulos posibles.

Método	Nº muestras	Longitud de vector		
		8 bits	16 bits	32 bits
Aproximar/Interpolar	13	70,05%	76,88%	76,97%
	25	34,60%	37,46%	37,56%
	37	21,92%	24,47%	24,57%
Interpolar/Seno (matlab)	13	3,13%	2,51%	2,51%
	25	1,71%	0,63%	0,59%
	37	1,81%	0,40%	0,26%
Taylor/Seno (matlab)			0,033%	

Tabla 6.8. Resumen de errores calculados

Los primeros diseños fiables se obtienen en los resultados de la interpolación. El más óptimo como es lógico se encuentra con 37 muestras y 32 bits, sin embargo, ya se sabe que el diseño de 32 bits no es sintetizable, por lo tanto, el diseño mediante la interpolación con 37 muestras y longitud de 16 bits, es el primer diseño realmente óptimo.

Finalmente, el diseño más apto se obtiene con la serie de Taylor donde se puede apreciar que el error desciende hasta las centésimas, consiguiendo así un resultado bastante aceptable.

6.4. Implementación en FPGA

En este apartado se detalla la implementación del diseño digital que realiza el cálculo del seno en una FPGA. Es necesario generar nuevos bloques que conecten el diseño con los dispositivos de interfaz presenentes en la placa.

6.4.1. Spartan-3E Starter Kit Board

La FPGA elegida pertenece a la familia Spartan-3E, concretamente es el dispositivo XC3S500E-FG320. Esta placa dispone de 50 MHz a bordo de reloj, Flash de 128Mbit, 16Mbit SPI flash, 64Mbyte DDR SDRAM, 4 canales de CAD, ADC de canal dual, LCD, codificador rotatorio, 8 conmutadores y leds, y conectores USB, VGA, Ethernet, FX2-J3, etc. Sin embargo, la placa no dispone de un display led de 7 segmentos, por tanto, se tiene que montar uno externo y conectárselo a la placa, para así poder visualizar los resultados. Además, debido a los pocos pines que tiene para conexiones exteriores, también se acopla una placa auxiliar que se adapta a un conector de la FPGA, para así habilitar un mayor número de pines [22].

En la siguiente imagen se marcan las partes principales que se han usado de la FPGA. Se puede ver la conexión de alimentación, la conexión vía USB, los *switches* utilizados para introducir el ángulo, los botones para la máquina de estados, y finalmente, el conector FX2-J3 que habilita 43 pines más de entrada/salida.

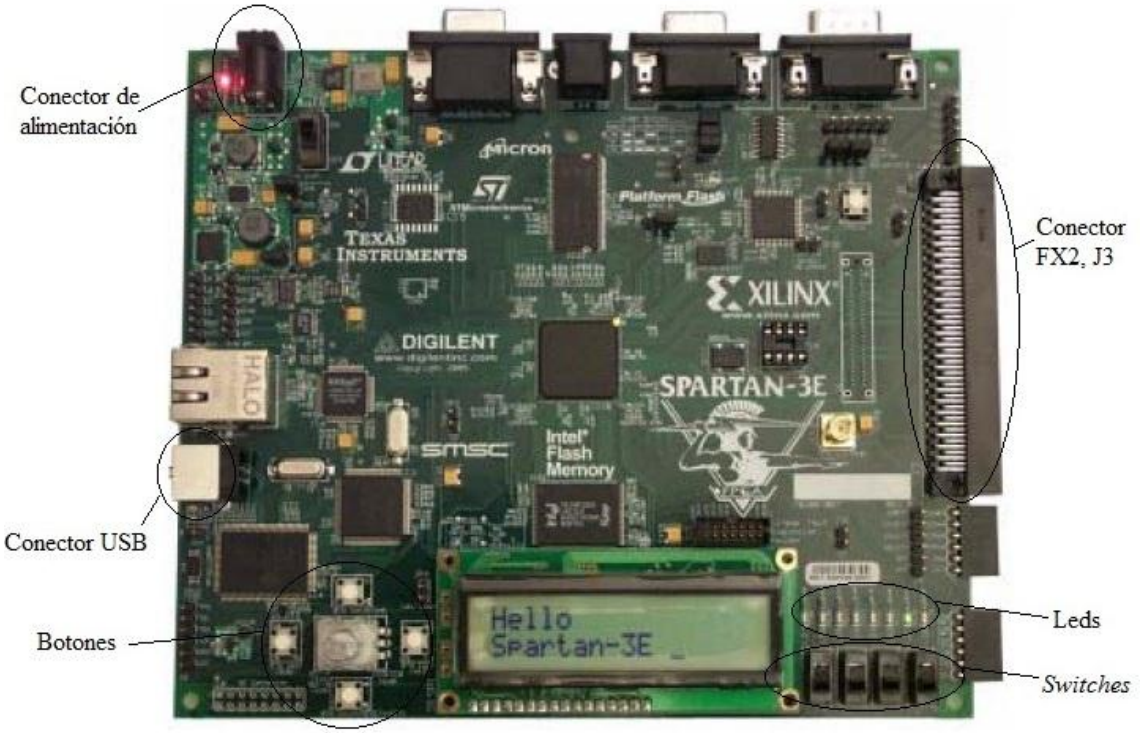


Figura 6.20. Spartan-3E XC3S500E

A la derecha de la imagen se puede ver el conector FX2-J3 donde se introduce la placa que habilita los 43 pines de entrada/salida.

Figura 6.21. Placa adaptación para pines

Finalmente se dispone de un display cuádruple de 7 segmentos donde se muestra el ángulo de entrada, y también el resultado del seno con tres decimales. Además, se añade otro display individual para mostrar el signo del seno hallado.

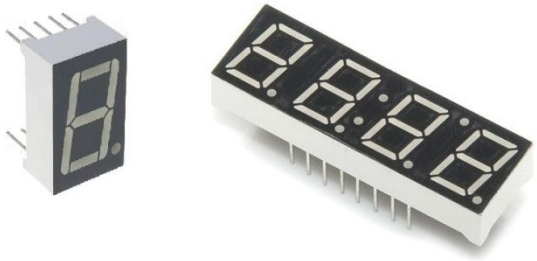


Figura 6.22. Displays individual (SA52-11EWA) y cuádruple (LM5644R-11N) de 7 segmentos

El display cuádruple solo tiene doce pines, lo que significa que simplemente se le puede pasar el valor de un dígito en código de 7 segmentos. Luego hay cuatro pines que seleccionan que dígitos muestran valor y cuáles no. Intuitivamente se puede pensar que si solo es posible mandarle un valor al display, no se pueden mostrar cuatro dígitos a la vez, por ejemplo: 0,579. Sin embargo, al estar trabajando a una frecuencia de MHz, lo que se hace es que en cada flanco de reloj se activa un dígito diferente y se le pasa su valor correspondiente, y así sucesivamente creando una secuencia. Debido a que el ojo humano solo es capaz de percibir alrededor de 25 imágenes por segundo, no se puede apreciar el parpadeo continuo que se produce en el display, y se ve como si estuviera continuamente activado y mostrando los valores correspondientes [23][24].

Una vez montado todo el conjunto, el resultado final de él se puede apreciar en la siguiente imagen.

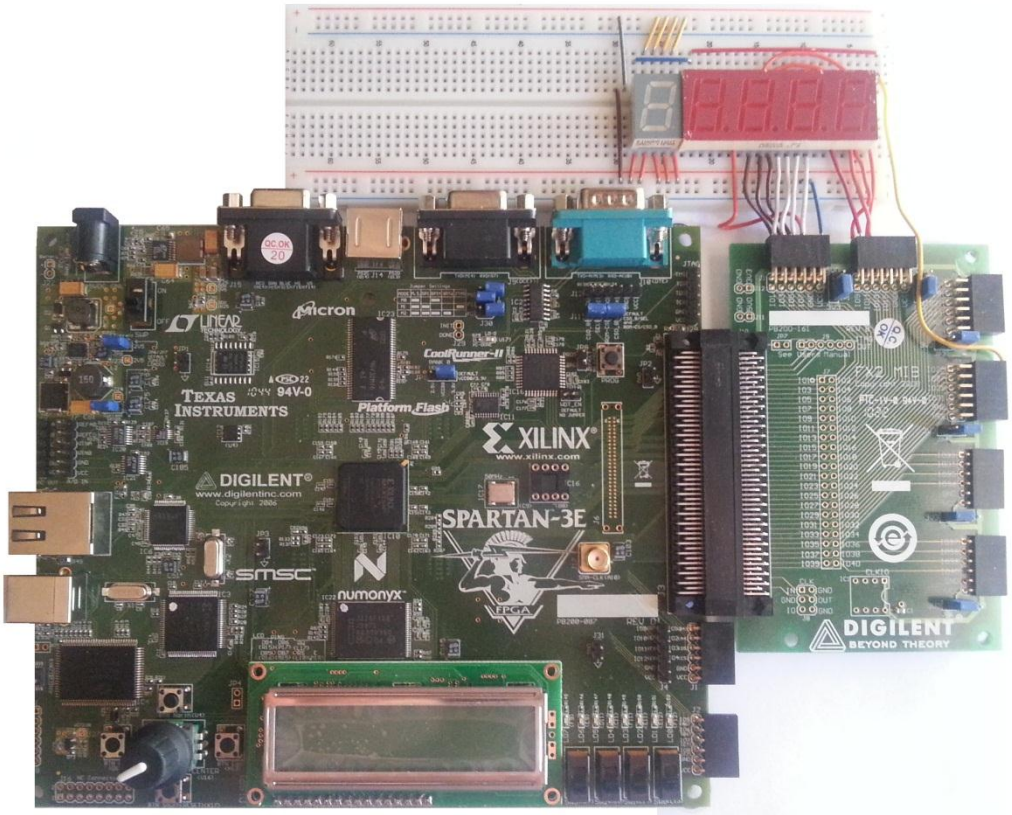


Figura 6.23. Montaje final Spartan-3E XC3S500E

6.4.2. Diagrama de bloques

Para llevar a cabo la implementación se crean varios bloques más para poder interactuar con los periféricos existentes. Se crea un bloque principal que se rige con una máquina de estados. Con este bloque se puede introducir el ángulo y también sacar los dígitos correspondientes por el display. El otro bloque necesario de añadir es el de un *contador*, que además como se tienen que usar varios, se realiza uno genérico. Finalmente el bloque que queda es el propio del diseño de la función seno mediante la serie de Taylor.

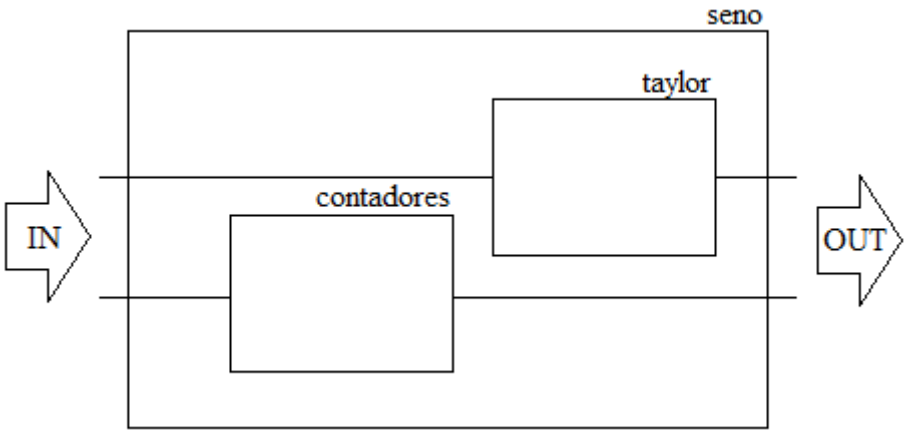


Figura 6.24. Diagrama de bloques de la implementación

En los siguientes apartados se detalla detenidamente el funcionamiento de cada bloque, exceptuando el de Taylor, ya que este bloque es en sí el diseño de nuestro proyecto, que ha sido explicado en los capítulos anteriores.

6.4.3. Diseño del bloque contador

El *contador* como su propio nombre indica lo único que hace es contar. Debido a la necesidad de instanciar varios, se diseña uno genérico, en el cuál se puede variar la dimensión de cuenta, para así no ocupar memoria innecesaria.

```
entity contador is
   generic(ancho : integer := 8);
   port (
      clk    : in std_logic;
      reset  : in std_logic;
      clear  : in std_logic;
      enable : in std_logic;
      cuenta : out std_logic_vector (ancho-1 downto 0));
end contador;
```

Figura 6.25. Entidad contador

Una vez vistos los puertos que tiene el *contador*, se realiza el diseño de la arquitectura de este.

```
architecture contador of contador is
begin
 process(clk, reset)
   variable count: std_logic_vector (ancho-1 downto 0);
begin
   if reset = '1' then
      for i in 0 to ancho-1 loop
         count(i) := '0';
      end loop;
   elsif clk'event and clk = '1' then
      if enable = '1' then
         if clear = '1' then
            for i in 0 to ancho-1 loop
               count(i) := '0';
            end loop;
         else
            count := count + '1';
         end if;
      end if;
   end if;
   cuenta <= count;
 end process;
end contador;
```

Figura 6.26. Arquitectura de la entidad contador

Como se puede ver el diseño es bastante intuitivo, donde hay un *reset* para inicializar el contador a cero, un *enable* para activarlo y finalmente un *clear* para limpiarlo cuando sea necesario.

6.4.4. Diseño del bloque seno

El bloque *seno* es el de mayor nivel de abstracción, tal y como se puede apreciar en la *figura 6.24*. En ella se instancian dos componentes, que son *taylor* y *contador*. Este bloque tiene dos partes diferenciadas, una es la máquina de estados bajo la que se rige el proceso, y la otra, es el anti-rebotes creado para poder usar los botones de la FPGA sin que estos produzcan ningún tipo de rebote.

A continuación se ven los puertos de la entidad seno, los cuales estan unidos a distintos botones, leds, pines, etc. de la placa.

```
entity seno is
   port(
      clk            : in std_logic;                          --reloj
      reset          : in std_logic;                          --reset
      int,cap,res    : in std_logic;                          --botones
      s0,s1,s2,s3    : in std_logic;                          --switches
      led_ini, led_1, led_2, led_3, led_ang, led_sen : out std_logic; --leds
      signo          : out std_logic;                         --display
      digito_7seg    : out std_logic_vector(7 downto 0);      --display
      digitos        : out std_logic_vector(3 downto 0));     --display
end seno;
```

Figura 6.27. Entidad seno

Al principio de la arquitectura del seno se definen e instancian los dos componentes necesarios, además de crear las señales pertinentes.

```
architecture a of seno is
 component Taylor is
  port(
     clk       : in  std_logic;
     reset     : in  std_logic;
     angulos   : in  std_logic_vector(8 downto 0);
     bus_datos : out std_logic_vector(15 downto 0));
 end component;
 component contador is
  generic(Ancho : integer := 8);
  port(
     Clk     : in std_logic;
     Reset   : in std_logic;
     Clear   : in std_logic;
     Enable  : in std_logic;
     Cuenta  : out std_logic_vector (Ancho-1 downto 0));
 end component;

--Señales para el conformador de pulsos del anti-rebotes
signal Q0,Q1,Sint,Sint2,Q2,Q3,Scap,Scap2,Q4,Q5,Sres,Sres2: std_logic;

--Señales para los contadores
signal clear,enable,clear0,enable0,clear1,enable1 : std_logic;
signal retardo: std_logic_vector(23 downto 0);
signal retraso0,retraso1: std_logic_vector(17 downto 0);

--Señales para la máquina de estados
type estados is (inicio, ang1, ang2, ang3, ang_valido, ang_display, dig1,
dig2, dig3, dig4);
signal actual, siguiente: estados;

--Señales para almacenar el valor del angulo y de su seno
signal ang: std_logic_vector(8 downto 0);
signal sen_dec: std_logic_vector(15 downto 0);

--Otras señales intermedias
signal num1,num2,num3 : integer range 0 to 9;
signal sen_7seg : std_logic_vector(31 downto 0);
signal ang_int1, ang_int2, ang_int3 : std_logic_vector(3 downto 0);
signal ang_comp: std_logic_vector(8 downto 0);

begin

--Instanciamos los componentes.
--Tenemos tres contadores. Uno para el anti-rebotes, y los otros dos nos
--servirán para sacar los valores del angulo y del seno por el display
taylor0    : taylor port map(clk,reset,ang,sen_dec);
antirebotes: contador generic map(24) port map(clk, reset, clear, enable,
retardo );
retraso_ang: contador generic map(18) port map(clk, reset, clear0, enable0,
retraso0);
retraso_sen: contador generic map(18) port map(clk, reset, clear1, enable1,
retraso1);
```

Figura 6.28. Arquitectura de la entidad seno

- Anti-rebotes

Los botones de la FPGA son mecánicos, y por tanto, al pulsarlos pueden producirse rebotes, sobre todo debido a la alta frecuencia usada en la implementación. En la *figura 6.29* se puede ver como en la primera y tercera pulsación del botón se producen rebotes, mientras que en la segunda no se produce ninguno.

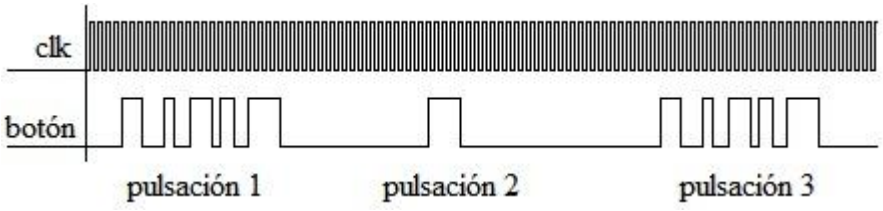


Figura 6.29. Rebotes

Por ejemplo, si se estuviera manejando la nave de un videojuego, al pulsar el botón de mover la nave a la izquierda una vez, esta se movería varías posiciones debido a los rebotes. Estos rebotes son imprevisibles y no siempre se producen, sin embargo, hay que buscar una manera de solucionarlo. Para ello se ha creado un anti-rebotes, el cual lo que hace es detectar si ha habido flanco de reloj y a su vez si ha pasado un cierto tiempo entre pulsación y pulsación.

Para detectar el flanco de reloj se ha diseñado un conformador de pulsos, tal y como se puede ver a continuación.

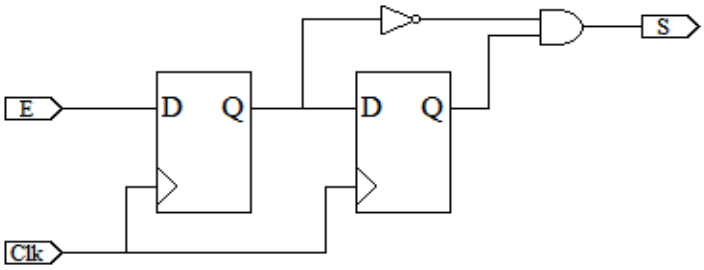


Figura 6.30. Conformador de pulsos

El conformador lo que hace es detectar si ha habido un flanco de reloj, en concreto un flanco de bajada, y cuando es así la salida se pone a '1'. La lógica combinacional de la salida es $S = \overline{Q0} \cdot Q1$.

Además se añade un retardo, donde tiene que pasar un cierto tiempo entre pulsación y pulsación para que se haga caso a la pulsación del botón, haciendo que los rebotes que se producen en medio sean ignorados. El retardo es de 100 ms, se ha calculado multiplicando el periodo de reloj (20 ns) por el valor de cuenta que devuelve el contador (5.000.000).

Una vez hechos estos dos pasos, se actualiza la *figura 6.29*, donde ahora aparece una señal que se activa solo cuando se cumplan ambas condiciones, es decir, que *S = '1'* y que el retardo sea mayor de 100 ms.

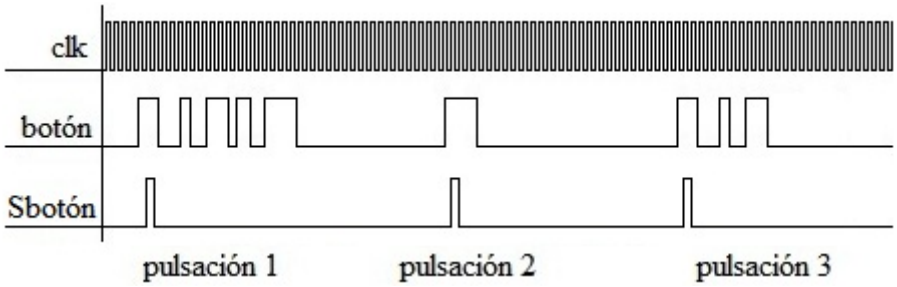


Figura 6.31. Anti-rebotes

El diseño en VHDL del anti-rebotes se hace siguiendo los pasos expuestos. Como en nuestro proceso se usan tres botones, se diseña lo anteriormente explicado tres veces.

```
process(clk,reset)
   constant cte_retardo : integer := 5000000; --100ms
begin
   if (reset = '1') then
      enable <= '0';
      clear <= '0';
   elsif clk'event and clk = '1' then   --En cada flanco voy almacenando
      Q0 <= int;                        --los estados de los botones
      Q1 <= Q0;
      Q2 <= cap;
      Q3 <= Q2;
      Q4 <= res;
      Q5 <= Q4;
      --Lógica del temporizador que hace de anti-rebote
      if retardo > cte_retardo then
         enable <= '0';
      else
         enable <= '1';
         clear <= '0';
      end if;
      --Lógica para activar la señal de que se ha pulsado un botón y a su
      --vez resetear el temporizador si hemos pasado ya los 100 ms
      if Sint2 = '1' and retardo > cte_retardo then
         Sint <= '1';
         Clear <= '1';
         Enable <= '1';
      elsif Scap2 = '1' and retardo > cte_retardo then
         Scap <= '1';
         Clear <= '1';
         Enable <= '1';
      elsif Sres2 = '1' and retardo > cte_retardo then
         Sres <= '1';
         Clear <= '1';
         Enable <= '1';
      else
         Sint <= '0';
         Scap <= '0';
         Sres <= '0';
      end if;
   end if;
end process;
--Lógica combinacional del conformador de pulsos
Sint2 <= not(Q0) and Q1;
Scap2 <= not(Q2) and Q3;
Sres2 <= not(Q4) and Q5;
```

Figura 6.32. Diseño VHDL del anti-rebotes

- Máquina de estados

La máquina de estados diseñada se divide en dos partes. La primera consiste en varios estados que sirven para introducir el ángulo y mostrarlo, y la segunda parte es sacar el valor del seno de dicho ángulo.

A continuación se dibuja el diagrama de estados de una máquina de Moore simplificada. No se ha introducido todo lo que hay en cada estado, ya que no se entendería bien. Por ejemplo, se omiten los leds que esten apagados en cada estado.

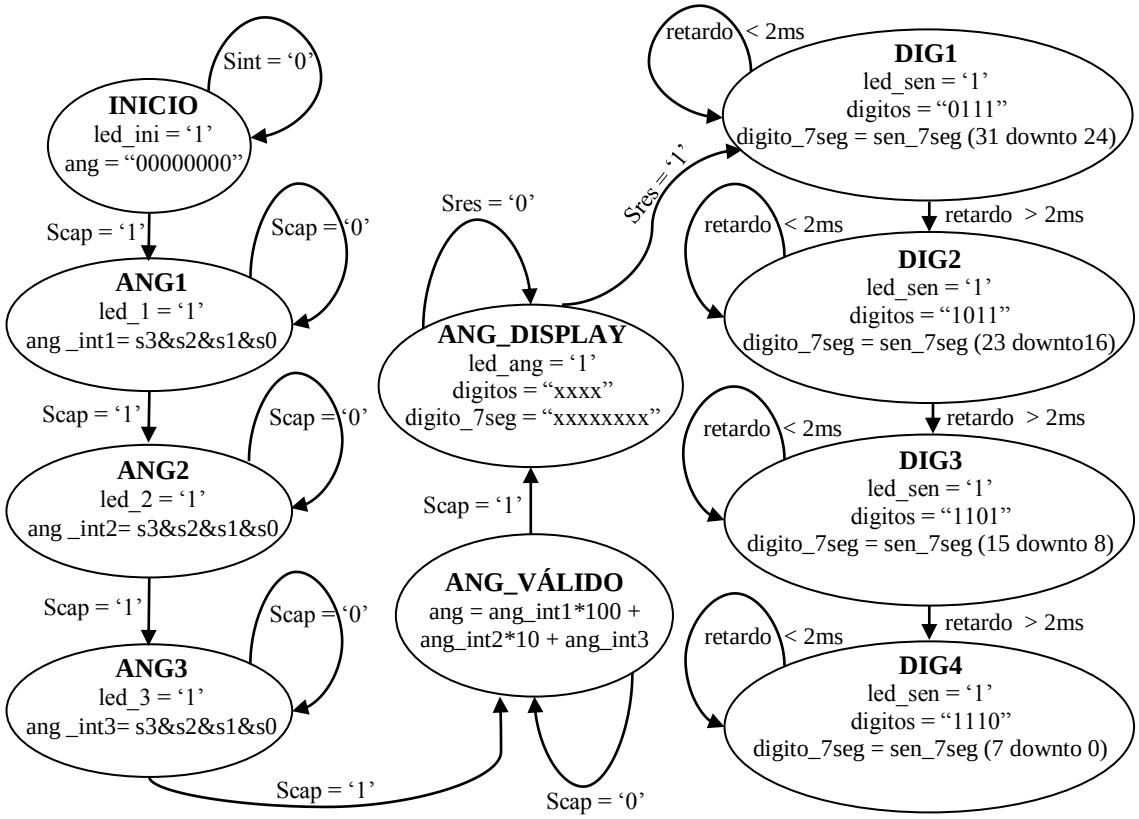


Figura 6.33. Máquina de estados de Moore simplificada

El único estado cuyo funcionamiento podría quedar más en el aire es el de *ANG_DISPLAY*, en este estado se saca por display el valor del ángulo introducido, de ahí que aparezcan las *'X'*. Se crea una secuencia de IFs con los que se va mostrando un valor u otro dependiendo del valor de cuenta que lleve un temporizador que hay especialmente para ello. Este estado se verá mejor cuando detallemos la descripción en VHDL.

En las siguientes líneas se muestra una breve descripción de lo que realiza cada estado. Por ejemplo, introduciendo el ángulo de 152º el seno sería 0.469.

- INICIO: Inicialización del ángulo a 0º.
- ANG1: Introdución del primer dígito del ángulo, que es uno.
- ANG2: Introdución del segundo dígito del ángulo, que es cinco.
- ANG3: Introdución del segundo dígito del ángulo, que es dos.
- ANG_VALIDO: Juntar los tres dígitos para formar el ángulo.
- ANG_DISPLAY: Mostrar el ángulo por el display.
- DIG1: Mostrar el dígito uno del seno, que es cero.
- DIG2: Mostrar el dígito dos del seno, que es cuatro.
- DIG3: Mostrar el dígito tres del seno, que es seis.
- DIG4: Mostrar el dígito cuatro del seno, que es nueve.

Cabe destacar que los valores de s3, s2, s1 y s0 son los *switches* de la placa, gracias a los cuales se introducen los dígitos del ángulo mediante código binario.

La máquina de estados plasmada en la *figura 6.33* queda totalmente implementada con todo nivel de detalle en la arquitectura de la entidad seno.

```
process(Clk, Reset)
begin
   if reset='1' then
      actual <= inicio;
   elsif clk'event and clk = '1' then
      actual <= siguiente;
   end if;
end process;
process(actual,Sint,Scap,Sres,retraso0,retraso1,ang)
   constant cte_retraso0 : integer := 100000;
   constant cte_retraso1 : integer := 100000;
begin
   case actual is
      when inicio =>
         ang <= "000000000";
         led_ini <= '1';   led_1   <= '0';     led_2   <= '0';
         led_3   <= '0';   led_ang <= '0';     led_sen <= '0';
         if (Sint = '1') then
            siguiente <= ang1;
         else
            siguiente <= inicio;
         end if;
      when ang1 =>
         ang_int1 <= s3 & s2 & s1 & s0;
         led_ini <= '0';   led_1   <= '1';     led_2   <= '0';
         led_3   <= '0';   led_ang <= '0';     led_sen <= '0';
         if(Scap = '1') then
            siguiente <= ang2;
         else
            siguiente <= ang1;
         end if;
      when ang2 =>
         ang_int2 <= s3 & s2 & s1 & s0;
         led_ini <= '0';   led_1   <= '0';     led_2   <= '1';
         led_3   <= '0';   led_ang <= '0';     led_sen <= '0';
         if (Scap = '1') then
            siguiente <= ang3;
         else
            siguiente <= ang2;
         end if;
      when ang3 =>
         ang_int2 <= s3 & s2 & s1 & s0;
         led_ini <= '0';   led_1   <= '0';     led_2   <= '0';
         led_3   <= '1';   led_ang <= '0';     led_sen <= '0';
         if (Scap = '1') then
            siguiente <= ang_valido;
         else
            siguiente <= ang3;
         end if;
      when ang_valido =>
         led_ini <= '0';   led_1   <= '0';     led_2   <= '0';
         led_3   <= '0';   led_ang <= '0';     led_sen <= '0';
         num1 <= conv_integer(ang_int1);
         num2 <= conv_integer(ang_int2);
         num3 <= conv_integer(ang_int3);
         ang  <= (ang_int1 * "1100100") + (ang_int2 * "1010") + (ang_int3);
         if (Scap = '1') then
            siguiente <= ang_display;
         else
            siguiente <= ang_valido;
         end if;
```

Figura 6.34. Diseño VHDL de la máquina de estados

En esta primera parte se ve que los primeros estados sirven para ir introduciendo el valor del ángulo dígito a dígito con los *switches*. Mientras que el último estado junta todos esos dígitos en la señal *ang*. Simplemente se hace multiplicando cada dígito por 100, 10 y 1 respectivamente, y sumándolo. Para así los dígitos pasar de ser unitarios a ser las centenas, decenas y unidades del ángulo correspondiente.

```
when ang_display =>
   led_ini <= '0';   led_1   <= '0';     led_2   <= '0';
   led_3   <= '0';   led_ang <= '1';     led_sen <= '0';
   enable0 <= '1';   clear0  <= '0';

   if (retraso0 < cte_retraso0) then     --2ms
      digitos <= "1011";
      if (ang < "1100100") then    -- Si ang < 100° el 2° dígito
         digito_7seg <= "00000000";-- se desactiva
      else
         digito_7seg <= fdisplay_ang(num1);
      end if;
   elsif (retraso0 >= cte_retraso0 and retraso0 < 2*cte_retraso0) then
      digitos <="1101";
      if (ang < "1010") then       -- Si ang < 10° el 3° digito
         digito_7seg <= "00000000";-- se desactiva
      else
         digito_7seg <= fdisplay_ang(num2);
      end if;
   elsif(retraso0 >= 2*cte_retraso0 and retraso0 < 3*cte_retraso0)then
      digitos <="1110";
      digito_7seg <= fdisplay_ang(num3);
   else
      clear0 <= '1';
   end if;

   if (Sres = '1') then
      if ang > "010110100" and ang < "101101000" then  --(180°,360°)
         signo <= '0'; --negativo
      else
         signo <= '1'; --positivo
      end if;
      sen_7seg <= fdisplay_sen(sen_dec);
      enable1 <= '1';
      clear1  <= '1';
      siguiente <= dig1;
   else
      siguiente <= ang_display;
   end if;
```

Figura 6.35. Diseño VHDL de la máquina de estados II

La función del estado *ANG_DISPLAY* se basa en mostrar por display el ángulo introducido. Como el display solo puede recibir el valor de un dígito, se usa un temporizador para ir enseñando cada cierto tramo de tiempo (2 ms) un digito diferente. La señal *dígitos* es de cuatro bits, y estos indican qué dígitos del display estan activos y cuales no. Se activan a nivel bajo. Por ejemplo, si *dígitos* es igual a “1011”, indica que el dígito que se enciende es el segundo. La otra señal importante es el valor que recibe el dígito en sí, que se le pasa a través de *digito_7seg*, y para ello se usa la función *fdisplay_ang* que se explicará más adelante.

La última parte de este estado es la transición al siguiente. Al pulsar el botón de *res* se manda la orden de obtener el seno del ángulo introducido. Lo primero es evaluar el signo del seno y posteriormente pasar a código 7 segmentos (*fdisplay_sen)* el valor

del seno calculado mediante *taylor*. Al tener ya el resultado se activa el contador necesario para mostrar el valor del seno en los siguientes estados (DIG1, DIG2, DIG3, DIG4).

Debido a que en la máquina de estados queda un poco ambigua la función de este estado, se realiza un diagrama de bloques con su funcionamiento.

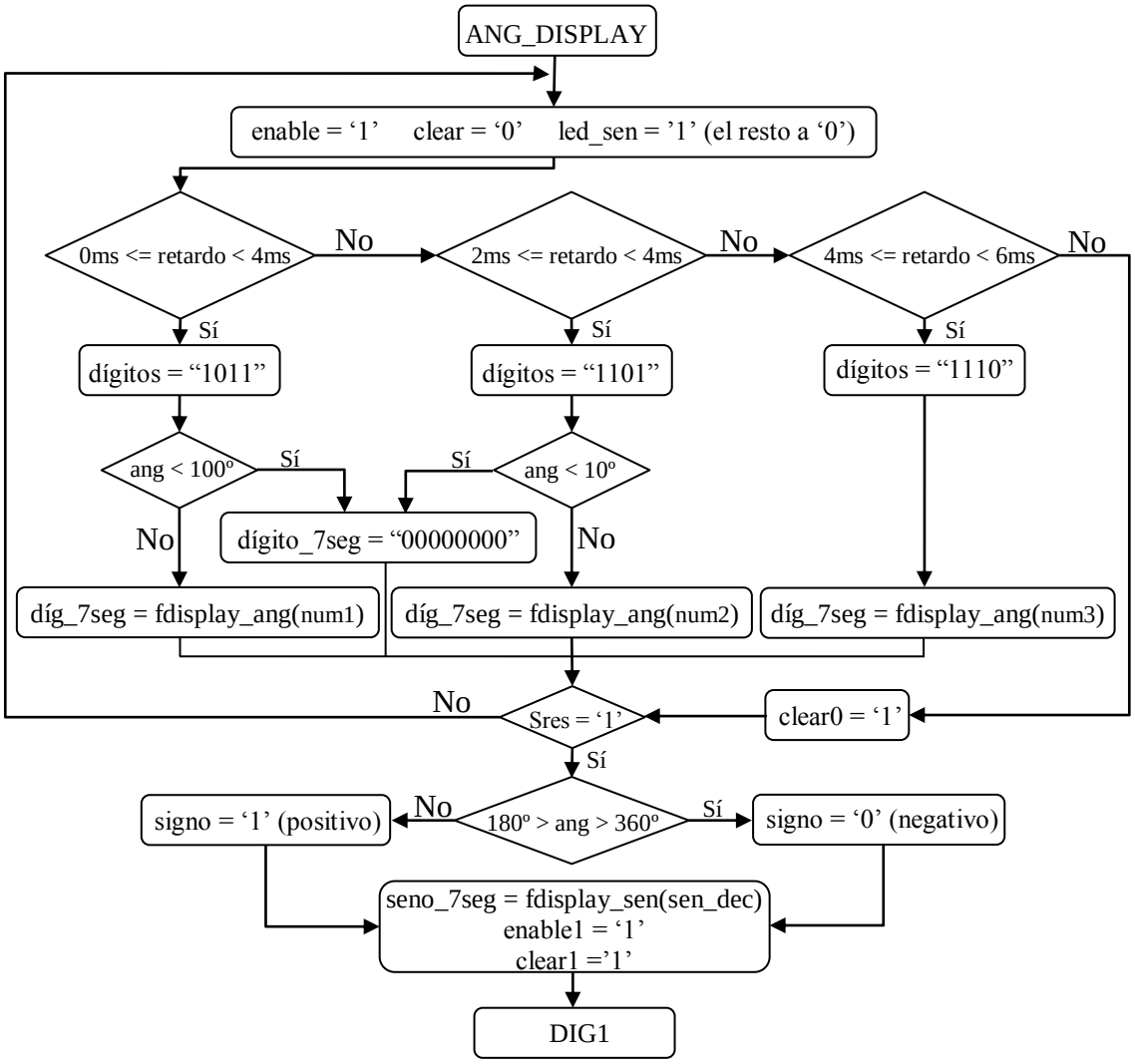


Figura 6.36. Diagrama de bloques del estado ANG_DISPLAY

Una vez mostrado el ángulo y pulsado el botón de *res* para calcular el seno, los siguientes estados se encargan de mostrar el resultado por el display. Para sacar el ángulo se utiliza un contador, mientras que ahora se realiza con una máquina de estados. Hay cuatro estados, donde cada estado representa un dígito del display.

```
      when dig1 =>
         led_ini <= '0'; led_1   <= '0'; led_2   <= '0';
         led_3   <= '0'; led_ang <= '0'; led_sen <= '1';
         clear1  <= '0';
         digito_7seg <= sen_7seg(31 downto 24);
         digitos <= "0111";
         if retraso1 > cte_retraso1 then
            clear1 <= '1';
            siguiente <= dig2;
         else
            siguiente <= dig1;
         end if;

      when(dig2) =>
         led_ini <= '0'; led_1   <= '0'; led_2   <= '0';
         led_3   <= '0'; led_ang <= '0'; led_sen <= '1';
         clear1  <= '0';
         digito_7seg <= sen_7seg(23 downto 16);
         digitos <= "1011";
         if retraso1 > cte_retraso1 then
            clear1 <= '1';
            siguiente <= dig3;
         else
            siguiente <= dig2;
         end if;

      when(dig3) =>
         led_ini <= '0'; led_1   <= '0'; led_2   <= '0';
         led_3   <= '0'; led_ang <= '0'; led_sen <= '1';
         clear1  <= '0';
         digito_7seg <= sen_7seg(15 downto 8);
         digitos <= "1101";
         if retraso1 > cte_retraso1 then
            clear1 <= '1';
            siguiente <= dig4;
         else
            siguiente <= dig3;
         end if;

      when(dig4) =>
         led_ini <= '0'; led_1   <= '0'; led_2   <= '0';
         led_3   <= '0'; led_ang <= '0'; led_sen <= '1';
         clear1  <= '0';
         digito_7seg <= sen_7seg(7 downto 0);
         digitos <= "1110";
         if retraso1 > cte_retraso1 then
            clear1 <= '1';
            siguiente <= dig1;
         else
            siguiente <= dig4;
         end if;
   end case;
end process;
```

Figura 6.37. Diseño VHDL de la máquina de estados III

Los cuatros estados son comunes y realizan la misma función, lo único que cambia es el dígito del display que encienden y el valor que toma cada dígito. Además se introduce otro contador para retardar el cambio entre estado y estado, ya que si se cambia de estado en cada flanco de reloj, los leds del display no funcionan bien, tal y como se puede apreciar en la *figura 6.39*. Esto seguramente se deba a que la frecuencia a la que trabaja la FPGA es mucho mayor a la frecuencia de conmutación que pueden soportar los leds del display.

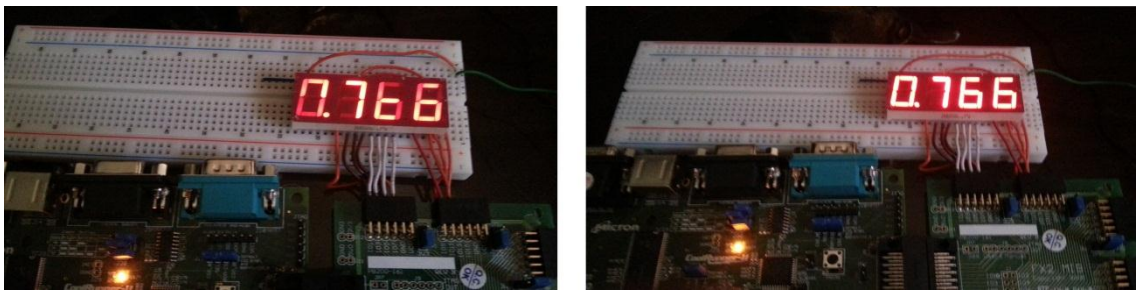


Figura 6.38. Resultado del seno de 50º sin temporización y con temporización de retardo

Como se puede ver la segunda imagen presenta una perfecta visualización del resultado, al contrario que sucede en la primera. A continuación se presentan varias imágenes para ver un ejemplo de los resultados obtenidos.

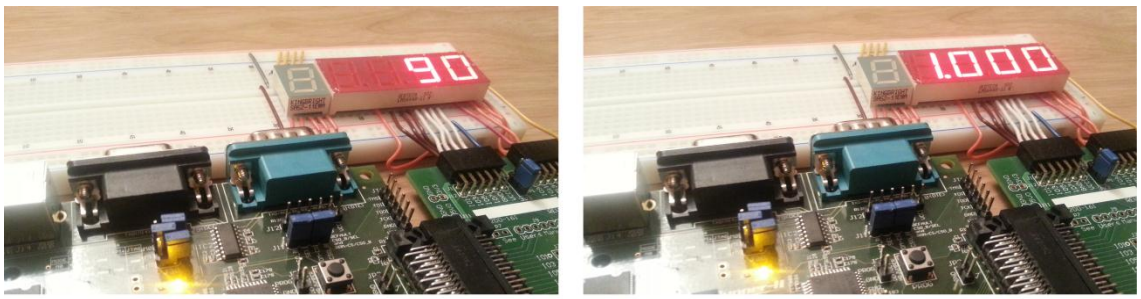


Figura 6.39 Ejemplo display: seno 90º = 1

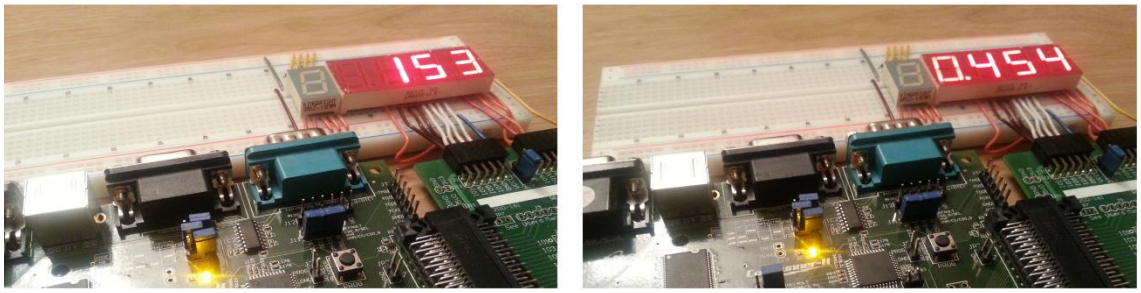


Figura 6.40. Ejemplo display: seno 153º = 0.454

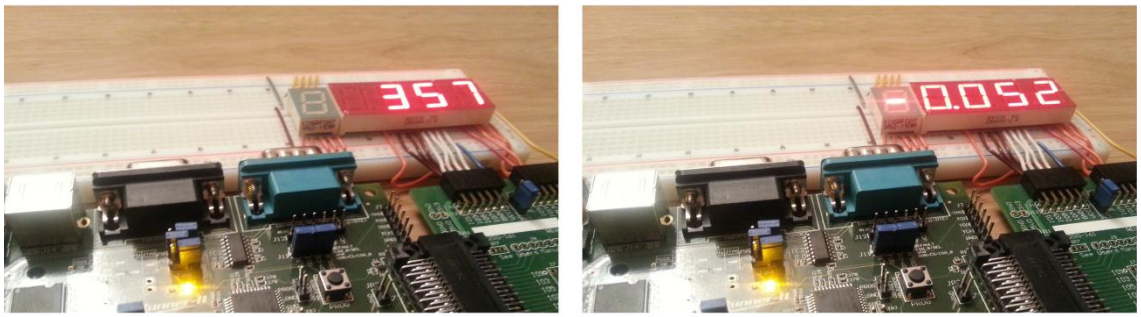


Figura 6.41. Ejemplo display: seno 357º = -0.052

6.4.5. Función *fdisplay_ang*

Esta función el cometido que tiene es pasar de código entero a 7 segmentos. Se le pasa como argumento un valor entre 0-9, y la función devuelve un vector de 8 bits, donde cada bit se corresponde con un led diferente, tal y como se puede ver en la siguiente imagen.

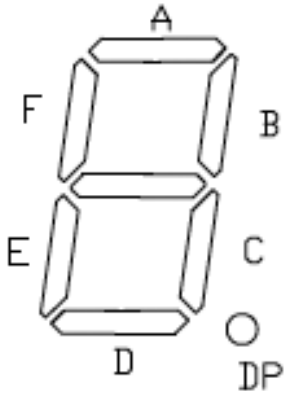


Figura 6.42. Esquema visual 7 segmentos

Por tanto, lo único que hay que hacer es una tabla donde a cada número le correspondan unos leds encendidos y otros apagados. Por ejemplo, para el siete, se tienen que encender A, B y C, y apagar el resto. Obteniendo así un vector cuyo valor sería "11100000". Se pone '1' (nivel alto) para activar los leds porque el display usado es de cátodo común, si se utilizase un display de ánodo común habría que activar los leds con '0' (nivel bajo) [23][24].

```
function fdisplay_ang (valor: integer) return std_logic_vector is
   variable salida: std_logic_vector(7 downto 0);
begin
   case valor is
      when(0) =>    salida :=  "11111100";
      when(1) =>    salida :=  "01100000";
      when(2) =>    salida :=  "11011010";
      when(3) =>    salida :=  "11110010";
      when(4) =>    salida :=  "01100110";
      when(5) =>    salida :=  "10110110";
      when(6) =>    salida :=  "10111110";
      when(7) =>    salida :=  "11100000";
      when(8) =>    salida :=  "11111110";
      when(9) =>    salida :=  "11110110";

      when others=> salida :=  "00000011";
   end case;
   return salida;
end;
```

Figura 6.43. Función fdisplay_ang

6.4.6. Función *fdisplay_sen*

Esta función convierte del formato decimal en que se obtiene el seno a código de 7 segmentos. El argumento que se le pasa es el vector de bits obtenido del bloque de *taylor*, que es el resultado del seno, se accede a una tabla mediante la presente función y esta encuentra el valor correspondiente convertido ya en código de 7 segmentos para pasárselo al display.

La tabla que contiene todo el rango de resultados posibles ha sido creada con una hoja de Excel. Primero se sacan todos los resultados posibles del diseño y se almacenan en un fichero *.txt*. Estos resultados se copian en una columna del Excel y se les separan los tres primeros decimales. Ahora ya teniendo los valores de los números de manera individual se pasan a código de 7 segmentos cada uno, y se juntan todos en un mismo vector. Un ejemplo con el número 0.172:

0. → "11111101"
1 → "01100000"
7 → "11100000"
2 → "11011010"

Si se juntan todos en un mismo vector se obtiene:

0.172 → "11111101_01100000_11100000_11011010"

Por tanto, para pasar cada número a los dígitos del display solo hay que acceder correctamente a las posiciones del vector que devuelve la función.

Debido a la repetitividad de la función, ya que simplemente es crear una tabla, se ha detallado solo un pequeño resumen con algunos ejemplos.

```
function fdisplay_sen (valor: std_logic_vector(15 downto 0)) return
std_logic_vector is
   variable salida: std_logic_vector(31 downto 0);
begin
   case valor is
      when("0000000000000000")=> salida:="11111101111111001111110011111100";
      when("0000000001000111")=> salida:="11111101111111000110000011100000";
      when("0000000010001111")=> salida:="11111101111111001111001001100110";
      …
      …
      …
      …
      …
      …
      …
      when("0000111111111110")=> salida:="11111101111101101111011011110110";
      when("0001000000000000")=> salida:="01100001111111001111110011111100";

      when others => salida := "00000000000000000000000000000000";
   end case;
   return salida;
end;
```

Figura 6.44. Función fdisplay_sen

6.4.7. Análisis de tiempos

En este apartado se analizan los tiempos de operación de los tres diseños realizados para la función seno. Se podrían comparar los resultados con lo que tarda en realizar un seno un computador, el matlab, una calculadora hp… sin embargo, se considera que no es una comparación del todo factible. En el caso de un ordenador, este, siempre esta realizando otro tipo de tareas en paralelo, aunque sean básicas. Mientras que una calculadora necesita de un microprocesador, el cual tiene un tiempo de desarrollo del seno mucho mayor que una FPGA. Es aquí cuando se demuestra la gran ventaja de usar una FPGA para estos propósitos, y es que, al poder reconfigurarla para tareas específicas, la velocidad del proceso es mucho mayor.

En la siguiente tabla se observan los distintos tiempos que tarda cada ciclo de reloj según el diseño, y consecuentemente, la frecuencia máxima a la que pueden funcionar.

Diseño	Tiempo mínimo de ciclo	Frecuencía máxima
Apróx-Interp. (8 bits)	4.915 ns	203.46 MHz
Apróx-Interp. (16 bits)	38.216 ns	26.167 MHz
Apróx-Interp. (32 bits)	-	-
Serie de Taylor	172.64 ns	5.792 MHz
Implementado	139.638 ns	7.161 MHz

Tabla 6.9. Resumen de tiempos

En el tercer diseño no aparece registro de tiempos, esto se debe a que no es sintetizable, y por tanto, tampoco se puede implementar en una FPGA. La diferencia de tiempos se debe sobre todo a la definición de cada diseño, ya que realizar una aproximación o interpolación es más sencillo que aplicar la serie de Taylor para la obtención de la función seno.

Capítulo 7

Conclusiones

El presente trabajo Fin de Grado planteó como objetivo principal la descripción de un módulo digital capaz de realizar la función seno mediante diferentes métodos, buscando siempre la mayor optimización y eficacia posibles, tanto en precisión, área y tiempo.

7.1. Acerca del uso de VHDL

- El lenguaje de descripción hardware VHDL permite diseñar, modelar y verificar un sistema desde un alto nivel de abstracción. Esta posibilidad hace que sea más fácil la compresión inicial del funcionamiento del sistema.

- Al estar basado en un estándar (IEEE std 1076-1987 y IEEE std 1076-199) puede utilizarse para reducir problemas de portabilidad.

- También permite poder validar cada componente de forma aislada al resto del sistema en el que esté incluido.

- Este lenguaje permite llevar a cabo simulaciones de los diseños y también sintetizar estos con ayuda de las herramientas adecuadas.

- Los componentes realizados en VHDL para un sistema pueden ser reutilizados para otro diseño diferente.

- El VHDL es una herramienta importante en la industria, pero también en la enseñanza, ya que es una herramienta útil para describir sistemas digitales.

7.2. Acerca de la exactitud de los resultados

El estudio analítico detallado de los errores cometidos en los distintos algoritmos utilizados para realizar la función seno era uno de los objetivos de este proyecto. Tras él, se pueden concluir varios aspectos:

- La utilización del método de aproximación es poco recomendable, puesto que los resultados obtenidos tienen un porcentaje de error excesivamente alto para poder trabajar en procesamiento de datos de control, cálculo, etc.
- El método de interpolación devuelve unos datos mucho más exactos, con un porcentaje de error bastante bajo. Además, se ha visto que cuanto mayor es el número de muestras disponibles, el porcentaje de error baja, como era de esperar.
- Finalmente, realizar un seno mediante la serie de Taylor es el método más exacto, cuyo porcentaje de error es prácticamente cero. También se ha podido observar que cuantos más términos se incluyan en la serie, mayor exactitud en los resultados.

A modo general, y exceptuando el método de aproximación, se ha observado que si se aumenta el ancho de palabra en los diseños, el error también disminuye. Esto se debe, a que se opera con mayor número de decimales, y por tanto, se esta más próximo del resultado exacto.

7.3. Líneas de trabajo futuras

Vistos los resultados concluidos en la realización del presente proyecto, aparecen dos líneas interesantes para futuros trabajos.

Uno de los puntos interesantes sería realizar un estudio más amplio del diseño mediante la serie de Taylor, concretamente variando el ancho de palabra utilizado en este método. Así se obtendrían distintos resultados, donde se vería cómo influyen más o menos bits, y en que proporción, en la exactitud del resultado.

El otro punto interesante, pero también bastante más complicado, sería cambiar el formato de almacenamiento de datos. Utilizando el estándar de IEEE para aritmética en coma flotante (IEEE 754).

Viendo estas dos ideas principales, la cumbre estaría en conseguir desarrollar la serie de Taylor con un ancho de palabra variable, y que además, se hiciese bajo el formato IEEE 754.

Capítulo 8

Presupuesto

En esta sección se presenta el presupuesto del trabajo. En él se contempla la duración de las tareas y un desglose de los distintos costes.

8.1. Tareas

Fase 1. Planificación

- Estudio de los distintos métodos para hallar la función seno.
  🕓 5 días
- Estudio de los distintos algoritmos para cada método.
  🕓 25 días

Fase 2. Desarrollo

- Análisis y diseño de cada algoritmo
  🕓 80 días
- Evaluación de resultados
  🕓 20 días
- Implementación en FPGA del algoritmo más óptimo
  🕓 30 días
- Evaluación de la aplicación
  🕓 5 días

Fase 3. Documentación

- Memoria del Trabajo Fin de Grado
  🕓 35 días
- Presentación audiovisual
  🕓 5 días

8.2. Recursos

Para la realización del presente trabajo han sido necesarios los siguientes recursos:

- **Recursos software:**
  - Licencia para herramienta para el diseño de circuitos digitales y su prototipado, Xilinx ISE Design Suite 13.2: **0 €**
  - Licencia Matlab: **6000 €**
  - Licencia Microsoft Office 2010: **200 €**
  - Licencia para simulador digital ISim 0.61xd: **0 €**
- **Recursos hardware:**
  - Ordenador portátil HP Pavilion g6: **800 €**
  - Spartan-3E Starter Kit Board: **150 €**
  - Placa auxiliar para pines: **20 €**
  - Protoboard y cables: **10 €**
  - Displays 7 segmentos: **10 €**
- **Recursos humanos:**
  - Tutor de proyecto
  - Desarrollador

  Las horas dedicadas al trabajo han sido una media de cuatro horas diarias, contemplando semanas de 5 días.
  - Coste de un ingeniero: **25 €/hora**

8.3. Resumen de costes

El coste total del sistema se presenta en la *tabla 8.1*:

CONCEPTO	IMPORTE
Recursos Humanos	20.500 €
Recursos Software	6.200 €
Recursos Hardware	990 €
Subtotal	27.690 €
(21 % IVA)	5.815 €
TOTAL	**33.505 €**

Tabla 8.1. Resumen del presupuesto

Anexos

Código de los diseños en VHDL

Diseño VHDL del seno mediante aproximación e interpolación (8 bits)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

use work.tabla_13.all;
use work.tabla_25.all;
use work.tabla_37.all;

entity seno is
   port(clk          : in std_logic;
      reset          : in std_logic;
      angulo         : in std_logic_vector (9 downto 0);
      tamaño         : in std_logic_vector (6 downto 0);
      bus_datos      : out std_logic_vector (7 downto 0);
      eleccion       : in std_logic);
end seno;

architecture funcion of seno is
begin
 process(clk,reset)
   variable bus_dir: integer range 0 to 361;
   variable y2,y1,y,x2,x1,x: integer;
   variable div,coc,i,j: integer;
 begin
   if reset = '1' then
      bus_datos <= "00000000";
   elsif clk'event and clk = '1' then
      bus_dir := conv_integer (angulo);
      case tamaño is
         when "0001101" =>
            if(angulo = angulos_13(bus_dir)) then
               bus_datos <= senos_13(bus_dir);
            elsif(eleccion = '0') then
               div := bus_dir;
```

```
            for i in 0 to 12 loop
               if (div > 29) then
                  div := div - 30;
               end if;
            end loop;
            y2 := bus_dir + (30 - div);
            bus_datos <= senos_13(y2);
         elsif(eleccion = '1') then
            div := bus_dir;
            for i in 0 to 12 loop
               if (div > 29) then
                  div := div - 30;
               end if;
            end loop;
            x  := bus_dir;
            x1 := bus_dir - div;
            x2 := bus_dir + (30 - div);
            y1 := conv_integer(senos_13(x1));
            y2 := conv_integer(senos_13(x2));

            div := (y2-y1)*(x2-x);
            coc := 0;
            i := 0;

            for i in 0 to 50 loop
               if(div > 30) then
                  coc := coc + 1;
                  div := div - 30;
               elsif(div < -30) then
                  coc := coc + 1;
                  div := div + 30;
               end if;
            end loop;

            if (div >= 0 and div <= 30) then
               y := y2 - coc;
            elsif (div >= -30 and div < 0) then
               y := y2 + coc;
            end if;

            bus_datos <= conv_std_logic_vector(y,8);
         end if;

      when "0011001" =>
         if(angulo = angulos_25(bus_dir)) then
            bus_datos <= senos_25(bus_dir);
         elsif(eleccion = '0') then
            div := bus_dir;
            for i in 0 to 24 loop
               if (div > 14) then
                  div := div - 15;
               end if;
            end loop;
            y2 := bus_dir + (15 - div);
            bus_datos <= senos_25(y2);
         elsif(eleccion = '1') then
            div := bus_dir;
            for i in 0 to 24 loop
               if (div > 14) then
                  div := div - 15;
               end if;
            end loop;
            x  := bus_dir;
            x1 := bus_dir - div;
            x2 := bus_dir + (15 - div);
```

```
            y1 := conv_integer(senos_25(x1));
            y2 := conv_integer(senos_25(x2));

            div := (y2-y1)*(x2-x);
            coc := 0;
            i := 0;

            for i in 0 to 25 loop
               if(div > 15) then
                  coc := coc + 1;
                  div := div - 15;
               elsif(div < -15) then
                  coc := coc + 1;
                  div := div + 15;
               end if;
            end loop;

            if (div > 0 and div < 15) then
               y := y2 - coc;
            elsif (div > -15 and div < 0) then
               y := y2 + coc;
            end if;

            bus_datos <= conv_std_logic_vector(y,8);
         end if;

      when "0100101" =>
         if(angulo = angulos_37(bus_dir)) then
            bus_datos <= senos_37(bus_dir);
         elsif(eleccion = '0') then
            div := bus_dir;
            for i in 0 to 36 loop
               if (div > 9) then
                  div := div - 10;
               end if;
            end loop;
            y2 := bus_dir + (10 - div);
            bus_datos <= senos_37(y2);
         elsif(eleccion = '1') then
            div := bus_dir;
            for i in 0 to 36 loop
               if (div > 9) then
                  div := div - 10;
               end if;
            end loop;
            x  := bus_dir;
            x1 := bus_dir - div;
            x2 := bus_dir + (10 - div);
            y1 := conv_integer(senos_37(x1));
            y2 := conv_integer(senos_37(x2));

            div := (y2-y1)*(x2-x);
            coc := 0;
            i := 0;

            for i in 0 to 16 loop
               if(div > 10) then
                  coc := coc + 1;
                  div := div - 10;
               elsif(div < -10) then
                  coc := coc + 1;
                  div := div + 10;
               end if;
            end loop;
```

```
            if (div >= 0 and div <= 10) then
               y := y2 - coc;
            elsif (div >= -10 and div < 0) then
               y := y2 + coc;
            end if;

            bus_datos <= conv_std_logic_vector(y,8);
         end if;

        when others =>
           bus_datos <= "00000000";
     end case;
   end if;
 end process;
end funcion;
```

Diseño VHDL del seno mediante aproximación e interpolación (16 bits)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

use work.tabla_13.all;
use work.tabla_25.all;
use work.tabla_37.all;

entity seno is
   port(clk          : in std_logic;
      reset          : in std_logic;
      angulo         : in std_logic_vector (9 downto 0);
      tamaño         : in std_logic_vector (6 downto 0);
      bus_datos      : out std_logic_vector (15 downto 0);
      eleccion       : in std_logic);
end seno;

architecture funcion of seno is
begin
 process(clk,reset)
   variable bus_dir: integer range 0 to 361;
   variable y2,y1,y,x2,x1,x: integer;
   variable div,coc,i,j: integer;
 begin
   if reset = '1' then
      bus_datos <= "00000000";
   elsif clk'event and clk = '1' then
      bus_dir := conv_integer (angulo);
      case tamaño is
         when "0001101" =>
            if(angulo = angulos_13(bus_dir)) then
               bus_datos <= senos_13(bus_dir);
            elsif(eleccion = '0') then
               div := bus_dir;
               for i in 0 to 12 loop
                  if (div > 29) then
                     div := div - 30;
                  end if;
               end loop;
               y2 := bus_dir + (30 - div);
               bus_datos <= senos_13(y2);
            elsif(eleccion = '1') then
               div := bus_dir;
```

```
            for i in 0 to 12 loop
               if (div > 29) then
                  div := div - 30;
               end if;
            end loop;
            x  := bus_dir;
            x1 := bus_dir - div;
            x2 := bus_dir + (30 - div);
            y1 := conv_integer(senos_13(x1));
            y2 := conv_integer(senos_13(x2));

            div := (y2-y1)*(x2-x);
            coc := 0;
            i := 0;

            for i in 0 to 5000 loop
               if(div > 30) then
                  coc := coc + 1;
                  div := div - 30;
               elsif(div < -30) then
                  coc := coc + 1;
                  div := div + 30;
               end if;
            end loop;

            if (div >= 0 and div <= 30) then
               y := y2 - coc;
            elsif (div >= -30 and div < 0) then
               y := y2 + coc;
            end if;
            bus_datos <= conv_std_logic_vector(y,8);
         end if;

      when "0011001" =>
         if(angulo = angulos_25(bus_dir)) then
            bus_datos <= senos_25(bus_dir);
         elsif(eleccion = '0') then
            div := bus_dir;
            for i in 0 to 24 loop
               if (div > 14) then
                  div := div - 15;
               end if;
            end loop;
            y2 := bus_dir + (15 - div);
            bus_datos <= senos_25(y2);
         elsif(eleccion = '1') then
            div := bus_dir;
            for i in 0 to 24 loop
               if (div > 14) then
                  div := div - 15;
               end if;
            end loop;
            x  := bus_dir;
            x1 := bus_dir - div;
            x2 := bus_dir + (15 - div);
            y1 := conv_integer(senos_25(x1));
            y2 := conv_integer(senos_25(x2));

            div := (y2-y1)*(x2-x);
            coc := 0;
            i := 0;

            for i in 0 to 2500 loop
               if(div > 15) then
                  coc := coc + 1;
                  div := div - 15;
```

```
               elsif(div < -15) then
                  coc := coc + 1;
                  div := div + 15;
               end if;
            end loop;

            if (div > 0 and div < 15) then
               y := y2 - coc;
            elsif (div > -15 and div < 0) then
               y := y2 + coc;
            end if;

            bus_datos <= conv_std_logic_vector(y,8);
         end if;

      when "0100101" =>
         if(angulo = angulos_37(bus_dir)) then
            bus_datos <= senos_37(bus_dir);
         elsif(eleccion = '0') then
            div := bus_dir;
            for i in 0 to 36 loop
               if (div > 9) then
                  div := div - 10;
               end if;
            end loop;
            y2 := bus_dir + (10 - div);
            bus_datos <= senos_37(y2);
         elsif(eleccion = '1') then
            div := bus_dir;
            for i in 0 to 36 loop
               if (div > 9) then
                  div := div - 10;
               end if;
            end loop;
            x  := bus_dir;
            x1 := bus_dir - div;
            x2 := bus_dir + (10 - div);
            y1 := conv_integer(senos_37(x1));
            y2 := conv_integer(senos_37(x2));

            div := (y2-y1)*(x2-x);
            coc := 0;
            i := 0;

            for i in 0 to 1600 loop
               if(div > 10) then
                  coc := coc + 1;
                  div := div - 10;
               elsif(div < -10) then
                  coc := coc + 1;
                  div := div + 10;
               end if;
            end loop;

            if (div >= 0 and div <= 10) then
               y := y2 - coc;
            elsif (div >= -10 and div < 0) then
               y := y2 + coc;
            end if;

            bus_datos <= conv_std_logic_vector(y,8);
         end if;

      when others =>
         bus_datos <= "00000000";
   end case;
```

```
   end if;
 end process;
end funcion;
```

Diseño VHDL del seno mediante aproximación e interpolación (32 bits)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

use work.tabla.all;

entity seno is
   port(clk         : in std_logic;
      reset         : in std_logic;
      angulo        : in std_logic_vector (9 downto 0);
      tamaño        : in std_logic_vector (6 downto 0);
      bus_datos     : out std_logic_vector (15 downto 0);
      eleccion      : in std_logic);
end seno;

architecture funcion of seno is
begin
 process(clk,reset)
   variable bus_dir: integer;
   variable cs: std_logic;
   variable tamaño2: integer;
   variable y2,y1,y,x2,x,z: integer;
 begin
   tamaño2 := conv_integer(tamaño);
   if reset = '1' then
      bus_datos <= "00000000000000000000000000000000";
   elsif clk'event and clk = '1' then
      case tamaño2 is

         when 13 =>
            bus_dir := 0;
            cs := '0';
            while (cs = '0' and bus_dir <= 13) loop
               if(angulo = angulos_13(bus_dir)) then
                  cs := '1';
                  bus_datos <= senos_13(bus_dir);
               elsif(eleccion = '0' and angulo < angulos_13(bus_dir)) then
                  cs := '1';
                  bus_datos <= senos_13(bus_dir);
               elsif(eleccion = '1' and angulo < angulos_13(bus_dir)) then
                  cs := '1';
                  y2 := conv_integer(senos_13(bus_dir));
                  y1 := conv_integer(senos_13(bus_dir-1));
                  x2 := conv_integer(angulos_13(bus_dir));
                  x := conv_integer(angulo);
                  z := (y2-y1)/30;
                  y := y2 - z*(x2-x);
                  bus_datos <= conv_std_logic_vector(y,32);
               else
                  cs := '0';
                  bus_dir := bus_dir + 1;
               end if;
            end loop;
```

```
        when 25 =>
           bus_dir := 0;
           cs := '0';
           while (cs = '0' and bus_dir <= 25) loop
              if(angulo = angulos_25(bus_dir)) then
                 cs := '1';
                 bus_datos <= senos_25(bus_dir);
              elsif(eleccion = '0' and angulo < angulos_25(bus_dir)) then
                 cs := '1';
                 bus_datos <= senos_25(bus_dir);
              elsif(eleccion = '1' and angulo < angulos_25(bus_dir)) then
                 cs := '1';
                 y2 := conv_integer(senos_25(bus_dir));
                 y1 := conv_integer(senos_25(bus_dir-1));
                 x2 := conv_integer(angulos_25(bus_dir));
                 x := conv_integer(angulo);
                 z := (y2-y1)/15;
                 y := y2 - z*(x2-x);
                 bus_datos <= conv_std_logic_vector(y,32);
              else
                 cs := '0';
                 bus_dir := bus_dir + 1;
              end if;
           end loop;

        when 37 =>
           bus_dir := 0;
           cs := '0';
           while (cs = '0' and bus_dir <= 37) loop
              if(angulo = angulos_37(bus_dir)) then
                 cs := '1';
                 bus_datos <= senos_37(bus_dir);
              elsif(eleccion = '0' and angulo < angulos_37(bus_dir)) then
                 cs := '1';
                 bus_datos <= senos_37(bus_dir);
              elsif(eleccion = '1' and angulo < angulos_37(bus_dir)) then
                 cs := '1';
                 y2 := conv_integer(senos_37(bus_dir));
                 y1 := conv_integer(senos_37(bus_dir-1));
                 x2 := conv_integer(angulos_37(bus_dir));
                 x := conv_integer(angulo);
                 z := (y2-y1)/10;
                 y := y2 - z*(x2-x);
                 bus_datos <= conv_std_logic_vector(y,32);
              else
                 cs := '0';
                 bus_dir := bus_dir + 1;
              end if;
           end loop;

        when others =>
           bus_dir := 0;
           cs := '0';
     end case;
   end if;
 end process;
end funcion;
```

Diseño VHDL del seno mediante la serie de Taylor

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

use work.tabla.all;
use work.funciones.all;

entity taylor is
   port(clk         : in std_logic;
        reset       : in std_logic;
        angulo      : in std_logic_vector (9 downto 0);
        bus_datos   : out std_logic_vector (15 downto 0));
end taylor;

architecture Behavioral of taylor is
   signal reg_bus_dir_a  : integer;
   signal reg_bus_dir_b  : integer;
   signal reg_bus_dir_c  : integer;
   signal reg_bus_dir_Ia : integer;
   signal reg_bus_dir_Ib : integer;
   signal reg_bus_dir_Ic : integer;
   signal reg_xa         : std_logic_vector (11 downto 0);
   signal reg_xb         : std_logic_vector (11 downto 0);
   signal reg_xc         : std_logic_vector (11 downto 0);
   signal reg_x2_int     : std_logic_vector (23 downto 0);
   signal reg_x3_int     : std_logic_vector (35 downto 0);
   signal reg_x3         : std_logic_vector (16 downto 0);
   signal reg_x5         : std_logic_vector (16 downto 0);
   signal reg_suma       : std_logic_vector (15 downto 0);
begin

 process(clk,reset)
   variable bus_dir, dir: integer;
   variable x: std_logic_vector(11 downto 0):= "000000000000";
 begin
   if reset = '1' then
      reg_xa <= "000000000000";
   elsif clk'event and clk = '1' then
      bus_dir := conv_integer(angulo);
      if bus_dir > 90 and bus_dir < 181 then
         dir := 180 - bus_dir;
      elsif bus_dir > 180 and bus_dir < 271 then
         dir := bus_dir - 180;
      elsif bus_dir > 270 and bus_dir < 361 then
         dir := 360 - bus_dir;
      elsif bus_dir > 360 then
         dir := bus_dir - 360;
      else
         dir := bus_dir;
      end if;
      reg_bus_dir_a  <= bus_dir;
      reg_bus_dir_Ia <= dir;
      x := angulos_radianes(dir);
      reg_xa <= x;
   end if;
 end process;

 process(clk,reset)
 begin
   if reset = '1' then
      reg_xb <= "000000000000";
```

```
  elsif clk'event and clk = '1' then
     reg_xb <= reg_xa;
     reg_bus_dir_Ib <= reg_bus_dir_Ia;
     reg_bus_dir_b <= reg_bus_dir_a;
  end if;
end process;

process(clk,reset)
begin
  if reset = '1' then
     reg_xc <= "000000000000";
  elsif clk'event and clk = '1' then
     reg_xc <= reg_xb;
     reg_bus_dir_Ic <= reg_bus_dir_Ib;
     reg_bus_dir_c <= reg_bus_dir_b;
  end if;
end process;

process(clk,reset)
  variable x2: std_logic_vector (23 downto 0;
  variable x3: std_logic_vector (35 downto 0;
begin
  if reset = '1' then
     reg_x2_int <= "000000000000000000000000";
     reg_x3_int <= "000000000000000000000000000000000000";
  elsif clk'event and clk = '1' then
     x2 := reg_xa * reg_xa;
     x3 := reg_xa * reg_xa * reg_xa;
     reg_x2_int <= x2;
     reg_x3_int <= x3;
  end if;
end process;

process(clk,reset)
  variable x3      : std_logic_vector (35 downto 0;
  variable x5      : std_logic_vector (59 downto 0;
  variable x3_aprox: std_logic_vector (16 downto 0);
  variable x5_aprox: std_logic_vector (16 downto 0);
  variable x3_div  : std_logic_vector (16 downto 0);
  variable x5_div  : std_logic_vector (16 downto 0);
begin
  if reset = '1' then
     reg_x3 <= "00000000000000000";
     reg_x5 <= "00000000000000000";
  elsif clk'event and clk = '1' then
     x3 := reg_x3_int;
     x5 := reg_x3_int * reg_x2_int;
     x3_aprox := aproximar ("000000000000000000000000" & x3);
     x5_aprox := aproximar (x5);
     x3_div := "000" & x3_aprox(16 downto 3);
     x5_div := "0000000" & x5_aprox(16 downto 7);
     reg_x3 <= x3_div;
     reg_x5 <= x5_div;
  end if;
end process;

process(clk,reset)
  variable suma1   : std_logic_vector (15 downto 0):= "0000000000000000";
  variable flag    : std_logic := '0';
begin
  if reset = '1' then
     reg_suma <= "0000000000000000";
  elsif clk'event and clk = '1' then
     suma1 := suma(reg_bus_dir_Ic,reg_xc,reg_x3,reg_x5);
     if reg_bus_dir_c > 180 and reg_bus_dir_c < 361 then
        suma1 := suma1 - '1';
```

```
        for i in 0 to 15 loop
          if suma1(i) = '1' and flag = '0' then
            flag := '1';
          elsif flag = '1' then
            suma1(i) := not suma1(i);
          end if;
        end loop;
      end if;
      reg_suma <= suma1;
   end if;
   suma1 := "0000000000000000";
 end process;
 bus_datos <= reg_suma;
end Behavioral;
```

Diseño VHDL de la implementación en FPGA del seno mediante la serie de Taylor

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.funciones_display.all;

entity seno is
   port(clk          : in std_logic;
      reset          : in std_logic;
      int,cap,res    : in std_logic;
      s0,s1,s2,s3    : in std_logic;
      led_ini, led_1, led_2, led_3, led_ang, led_sen : out std_logic;
      signo          : out std_logic;
      digito_7seg    : out std_logic_vector(7 downto 0);
      digitos        : out std_logic_vector(3 downto 0));
end seno;

architecture a of seno is
 component Taylor is
   port(clk          : in  std_logic;
      reset          : in  std_logic;
      angulos        : in  std_logic_vector(8 downto 0);
      bus_datos      : out std_logic_vector(15 downto 0));
 end component;
 component contador is
   generic(Ancho     : integer := 8);
   port(Clk          : in std_logic;
      Reset          : in std_logic;
      Clear          : in std_logic;
      Enable         : in std_logic;
      Cuenta         : out std_logic_vector (Ancho-1 downto 0));
 end component;
type estados is (inicio, ang1, ang2, ang3, ang_valido, ang_display, dig1,
dig2 ,dig3, dig4);
signal Q0,Q1,Sint,Sint2,Q2,Q3,Scap,Scap2,Q4,Q5,Sres,Sres2  : std_logic;
signal clear,enable,clear0,enable0,clear1,enable1          : std_logic;
signal retardo                          : std_logic_vector(22 downto 0);
signal retraso0,retraso1                : std_logic_vector(17 downto 0);
signal ang                              : std_logic_vector(8 downto 0);
signal sen_dec                          : std_logic_vector(15 downto 0);
signal sen_7seg                         : std_logic_vector(31 downto 0);
signal ang_int1, ang_int2, ang_int3     : std_logic_vector(3 downto 0);
signal ang_comp                         : std_logic_vector(8 downto 0);
signal num1,num2,num3                   : integer range 0 to 9;
signal actual, siguiente                : estados;
```

```
begin
taylor0    : taylor   port map(clk,reset,ang,sen_dec);
antirebotes: contador generic map(23)
                      port map(clk,reset,clear,enable,retardo);
retraso_ang: contador generic map(18)
                      port map(clk,reset,clear0,enable0,retraso0);
retraso_sen: contador generic map(18)
                      port map(clk,reset,clear1,enable1,retraso1);

 process(Clk, Reset)
 begin
   if reset='1' then
      actual <= inicio;
   elsif clk'event and clk = '1' then
      actual <= siguiente;
   end if;
 end process;

 process(actual,Sint,Scap,Sres,retraso0,retraso1,ang)
   constant cte_retraso0 : integer := 100000;--100000
   constant cte_retraso1 : integer := 100000;--100000
 begin
   case actual is
      when inicio =>
         ang <= "0000000000";
         led_ini <= '1'; led_1   <= '0'; led_2   <= '0';
         led_3   <= '0'; led_ang <= '0'; led_sen <= '0';
         if (Sint = '1') then
            siguiente <= ang1;
         else
            siguiente <= inicio;
         end if;

      when ang1 =>
         ang_int1 <= s3 & s2 & s1 & s0;
         led_ini <= '0'; led_1   <= '1'; led_2   <= '0';
         led_3   <= '0'; led_ang <= '0'; led_sen <= '0';
         if(Scap = '1') then
            siguiente <= ang2;
         else
            siguiente <= ang1;
         end if;

      when ang2 =>
         ang_int2 <= s3 & s2 & s1 & s0;
         led_ini <= '0'; led_1   <= '0'; led_2   <= '1';
         led_3   <= '0'; led_ang <= '0'; led_sen <= '0';
         if (Scap = '1') then
            siguiente <= ang3;
         else
            siguiente <= ang2;
         end if;

      when ang3 =>
         ang_int3 <= s3 & s2 & s1 & s0;
         led_ini <= '0'; led_1   <= '0'; led_2   <= '0';
         led_3   <= '1'; led_ang <= '0'; led_sen <= '0';
         if (Scap = '1') then
            siguiente <= ang_valido;
         else
            siguiente <= ang3;
         end if;
```

```
      when ang_valido =>
         led_ini <= '0'; led_1   <= '0'; led_2   <= '0';
         led_3   <= '0'; led_ang <= '0'; led_sen <= '0';
         num1 <= conv_integer(ang_int1);
         num2 <= conv_integer(ang_int2);
         num3 <= conv_integer(ang_int3);
         ang <= (ang_int1 * "1100100") + (ang_int2 * "1010") + (ang_int3);
         if (Scap = '1') then
            siguiente <= ang_display;
         else
            siguiente <= ang_valido;
         end if;

      when ang_display =>
         led_ini <= '0'; led_1   <= '0'; led_2   <= '0';
         led_3   <= '0'; led_ang <= '1'; led_sen <= '0';
         enable0 <= '1';
         clear0 <= '0';

         if (retraso0 < cte_retraso0) then
            digitos <= "1011";
            if (ang < "1100100") then
               digito_7seg <= "00000000";
            else
               digito_7seg <= fdisplay_ang(num1);
            end if;
         elsif(retraso0 >= cte_retraso0 and retraso0 < 2*cte_retraso0)then
            digitos <="1101";
            if (ang < "1010") then
               digito_7seg <= "00000000";
            else
               digito_7seg <= fdisplay_ang(num2);
            end if;
         elsif(retraso0 >= 2*cte_retraso0 and retraso0 < 3*cte_retraso0)then
            digitos <="1110";
            digito_7seg <= fdisplay_ang(num3);
         else
            clear0 <= '1';
         end if;

         if (Sres = '1') then
            if ang > "0101101000" and ang < "101101000" then
               signo <= '0'; --negativo
            else
               signo <= '1'; --positivo
            end if;
            sen_7seg <= fdisplay_sen(sen_dec);
            enable1 <= '1';
            clear1  <= '1';
            siguiente <= dig1;
         else
            siguiente <= ang_display;
         end if;

      when dig1 =>
         led_ini <= '0'; led_1   <= '0'; led_2   <= '0';
         led_3   <= '0'; led_ang <= '0'; led_sen <= '1';
         clear1  <= '0';
         digito_7seg <= sen_7seg(31 downto 24);
         digitos <= "0111";
         if retraso1 > cte_retraso1 then
            clear1 <= '1';
            siguiente <= dig2;
         else
            siguiente <= dig1;
         end if;
```

```
     when dig2 =>
        led_ini <= '0'; led_1   <= '0'; led_2   <= '0';
        led_3   <= '0'; led_ang <= '0'; led_sen <= '1';
        clear1  <= '0';
        digito_7seg <= sen_7seg(23 downto 16);
        digitos <= "1011";
        if retraso1 > cte_retraso1 then
           clear1 <= '1';
           siguiente <= dig3;
        else
           siguiente <= dig2;
        end if;

     when dig3 =>
        led_ini <= '0'; led_1   <= '0'; led_2   <= '0';
        led_3   <= '0'; led_ang <= '0'; led_sen <= '1';
        clear1  <= '0';
        digito_7seg <= sen_7seg(15 downto 8);
        digitos <= "1101";
        if retraso1 > cte_retraso1 then
           clear1 <= '1';
           siguiente <= dig4;
        else
           siguiente <= dig3;
        end if;

     when dig4 =>
        led_ini <= '0'; led_1   <= '0'; led_2   <= '0';
        led_3   <= '0'; led_ang <= '0'; led_sen <= '1';
        clear1  <= '0';
        digito_7seg <= sen_7seg(7 downto 0);
        digitos <= "1110";
        if retraso1 > cte_retraso1 then
           clear1 <= '1';
           siguiente <= dig1;
        else
           siguiente <= dig4;
        end if;
  end case;
end process;

process(clk,reset)
  constant cte_retardo : integer := 5000000; --5000000 --5
begin
  if (reset = '1') then
     enable <= '0';
     clear <= '0';
  elsif clk'event and clk = '1' then
     Q0 <= int;
     Q1 <= Q0;
     Q2 <= cap;
     Q3 <= Q2;
     Q4 <= res;
     Q5 <= Q4;
     if retardo > cte_retardo then
        enable <= '0';
     else
        enable <= '1';
        clear <= '0';
     end if;
     if Sint2 = '1' and retardo > cte_retardo then
        Sint <= '1';
        Clear <= '1';
        Enable <= '1';
     elsif Scap2 = '1' and retardo > cte_retardo then
        Scap <= '1';
```

```
        Clear <= '1';
        Enable <= '1';
     elsif Sres2 = '1' and retardo > cte_retardo then
        Sres <= '1';
        Clear <= '1';
        Enable <= '1';
     else
        Sint <= '0';
        Scap <= '0';
        Sres <= '0';
     end if;
   end if;
 end process;
 Sint2 <= not(Q0) and Q1;
 Scap2 <= not(Q2) and Q3;
 Sres2 <= not(Q4) and Q5;
end a;
```

Diseño VHDL del contador utilizado en la implementación

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity contador is
   generic(ancho : integer := 8);
   port (clk  : in std_logic;
      reset   : in std_logic;
      clear   : in std_logic;
      enable  : in std_logic;
      cuenta  : out std_logic_vector (ancho-1 downto 0));
end contador;

architecture contador of contador is
begin
 process(clk, reset)
   variable count: std_logic_vector (ancho-1 downto 0);
 begin
   if reset = '1' then
      for i in 0 to ancho-1 loop
         count(i) := '0';
      end loop;
   elsif clk'event and clk = '1' then
      if enable = '1' then
         if clear = '1' then
            for i in 0 to ancho-1 loop
               count(i) := '0';
            end loop;
         else
            count := count + '1';
         end if;
      end if;
   end if;
   cuenta <= count;
 end process;
end contador;
```

Bancos de pruebas de los diseños

Banco de pruebas del diseño del seno aproximado e interpolado con 8 bits

```
LIBRARY ieee;
use std.textio.all;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;

ENTITY tb2_seno IS
END tb2_seno;

ARCHITECTURE behavior OF tb2_seno IS
 COMPONENT seno
  PORT(clk          : IN  std_logic;
       reset        : IN  std_logic;
       angulo       : IN  std_logic_vector(9 downto 0);
       tamaño       : IN  std_logic_vector(6 downto 0);
       bus_datos    : OUT std_logic_vector(7 downto 0);
       eleccion     : IN  std_logic);
 END COMPONENT;
   --Inputs
   signal clk       : std_logic := '0';
   signal reset     : std_logic := '0';
   signal angulo    : std_logic_vector(9 downto 0) := (others => '0');
   signal tamaño    : std_logic_vector (6 downto 0);
   signal eleccion  : std_logic := '0';
   --Outputs
   signal bus_datos : std_logic_vector(7 downto 0);
   --Declaracion de fichero
   FILE output_file: TEXT is out "H:\Users\Ruben\Desktop\outfile.txt";
BEGIN
   -- Instantiate the Unit Under Test (UUT)
   uut: seno PORT MAP (clk => clk, reset => reset, angulo => angulo,
   tamaño => tamaño, bus_datos => bus_datos, eleccion => eleccion);
   -- Clock process definitions
   clk_process :process
   begin
       clk <= '0'; wait for 10 ns;
       clk <= '1'; wait for 10 ns;
   end process;
   stim_proc: process
       variable linea: line;
       variable resultado: integer;
   begin
       eleccion <= '1';    -- '0':Aproximacion -- '1':Interpolacion
       reset <= '1';
       wait for 10 ns;
       reset <= '0';
       tamaño <= "0001101";--"0001101"=13 | "0011001"=25 | "0100101”=37
       wait for 10 ns;
       angulo <= "0000000000";
       for i in 0 to 361 loop
              angulo <= angulo + '1';
              resultado := conv_integer(bus_datos);
              write(linea,resultado);
              writeline(output_file,linea);
              wait for 20 ns;
       end loop;
      wait;
   end process;
END;
```

Banco de pruebas del diseño del seno aproximado e interpolado con 16 bits

```
LIBRARY ieee;
use std.textio.all;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;

ENTITY tb2_seno IS
END tb2_seno;

ARCHITECTURE behavior OF tb2_seno IS
 COMPONENT seno
  PORT(clk          : IN  std_logic;
       reset        : IN  std_logic;
       angulo       : IN  std_logic_vector(9 downto 0);
       tamaño       : IN  std_logic_vector(6 downto 0);
       bus_datos    : OUT std_logic_vector(15 downto 0);
       eleccion     : IN  std_logic);
 END COMPONENT;
   --Inputs
   signal clk       : std_logic := '0';
   signal reset     : std_logic := '0';
   signal angulo    : std_logic_vector(9 downto 0) := (others => '0');
   signal tamaño    : std_logic_vector (6 downto 0);
   signal eleccion  : std_logic := '0';
   --Outputs
   signal bus_datos : std_logic_vector(15 downto 0);
   -- Clock period definitions
   constant clk_period : time := 10 ns;
   --Declaracion de fichero
   FILE output_file: TEXT is out "H:\Users\Ruben\Desktop\outfile.txt";
BEGIN
   -- Instantiate the Unit Under Test (UUT)
   uut: seno PORT MAP (clk => clk, reset => reset, angulo => angulo,
   tamaño => tamaño, bus_datos => bus_datos, eleccion => eleccion);
   -- Clock process definitions
   clk_process :process
   begin
       clk <= '0'; wait for 10 ns;
       clk <= '1'; wait for 10 ns;
   end process;
   stim_proc: process
       variable linea: line;
       variable resultado: integer;
   begin
       eleccion <= '1';    -- '0':Aproximacion -- '1':Interpolacion
       reset <= '1';
       wait for 10 ns;
       reset <= '0';
       tamaño <= "0001101";--"0001101"=13 | "0011001"=25 | "0100101”=37
       wait for 10 ns;
       angulo <= "0000000000";
       for i in 0 to 361 loop
              angulo <= angulo + '1';
              resultado := conv_integer(bus_datos);
              write(linea,resultado);
              writeline(output_file,linea);
              wait for 20 ns;
       end loop;
      wait;
   end process;
END;
```

Banco de pruebas del diseño del seno aproximado e interpolado con 32 bits

```
LIBRARY ieee;
use std.textio.all;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;

ENTITY tb2_seno IS
END tb2_seno;

ARCHITECTURE behavior OF tb2_seno IS
 COMPONENT seno
  PORT(clk          : IN  std_logic;
       reset        : IN  std_logic;
       angulo       : IN  std_logic_vector(9 downto 0);
       tamaño       : IN  std_logic_vector(6 downto 0);
       bus_datos    : OUT std_logic_vector(31 downto 0);
       eleccion     : IN  std_logic);
 END COMPONENT;
   --Inputs
   signal clk       : std_logic := '0';
   signal reset     : std_logic := '0';
   signal angulo    : std_logic_vector(9 downto 0) := (others => '0');
   signal tamaño    : std_logic_vector (6 downto 0);
   signal eleccion  : std_logic := '0';
   --Outputs
   signal bus_datos : std_logic_vector(31 downto 0);
   -- Clock period definitions
   constant clk_period : time := 10 ns;
   --Declaracion de fichero
   FILE output_file: TEXT is out "H:\Users\Ruben\Desktop\outfile.txt";
BEGIN
   -- Instantiate the Unit Under Test (UUT)
   uut: seno PORT MAP (clk => clk, reset => reset, angulo => angulo,
   tamaño => tamaño, bus_datos => bus_datos, eleccion => eleccion);
   -- Clock process definitions
   clk_process :process
   begin
       clk <= '0'; wait for 10 ns;
       clk <= '1'; wait for 10 ns;
   end process;
   stim_proc: process
       variable linea: line;
       variable resultado: integer;
   begin
       eleccion <= '1';    -- '0':Aproximacion -- '1':Interpolacion
       reset <= '1';
       wait for 10 ns;
       reset <= '0';
       tamaño <= "0001101";--"0001101"=13 | "0011001"=25 | "0100101”=37
       wait for 10 ns;
       angulo <= "0000000000";
       for i in 0 to 361 loop
              angulo <= angulo + '1';
              resultado := conv_integer(bus_datos);
              write(linea,resultado);
              writeline(output_file,linea);
              wait for 20 ns;
       end loop;
      wait;
   end process;
END;
```

Banco de pruebas del diseño del seno mediante la Serie de Taylor

```
LIBRARY ieee;
use std.textio.all;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;

ENTITY tb IS
END tb;

ARCHITECTURE behavior OF tb IS
 COMPONENT taylor
  PORT(clk          : IN  std_logic;
       reset        : IN  std_logic;
       angulo       : IN  std_logic_vector(9 downto 0);
       bus_datos    : OUT std_logic_vector(15 downto 0));
 END COMPONENT;
   --Inputs
   signal clk : std_logic := '0';
   signal reset : std_logic := '0';
   signal angulo : std_logic_vector(9 downto 0) := (others => '0');
   --Outputs
   signal bus_datos : std_logic_vector(15 downto 0);
   -- Clock period definitions
   constant clk_period : time := 10 ns;
   --Declaracion de fichero
   FILE output_file: TEXT is out "H:\Users\Ruben\Desktop\outfile.txt";
BEGIN
   -- Instantiate the Unit Under Test (UUT)
   uut: taylor PORT MAP (clk => clk, reset => reset, => angulo, bus_datos
  => bus_datos);
   -- Clock process definitions
   clk_process :process
   begin
       clk <= '0'; wait for 10 ns;
       clk <= '1'; wait for 10 ns;
   end process;
   -- Stimulus process
   stim_proc: process
       variable linea: line;
       variable ang,resultado: integer;
       variable grados: integer;
   begin
       reset <= '1';
       wait for 10 ns;
       reset <= '0';
       --Para interpretar el resultado se debe de ver las variables con  --
       signo (SIGNED) y el valor que sale
       --dividir entre 4096 (=2^12)
       angulo <= "0000000000";
       wait for 10 ns;
       for i in 0 to 363 loop
              ang := i;
              resultado := conv_integer(bus_datos);
              wait for 20 ns;
              write(linea,ang);
              writeline(output_file,linea);
              write(linea,bus_datos);
              writeline(output_file,linea);
              angulo <= angulo + '1';
       end loop;
      wait;
   end process;
END;
```

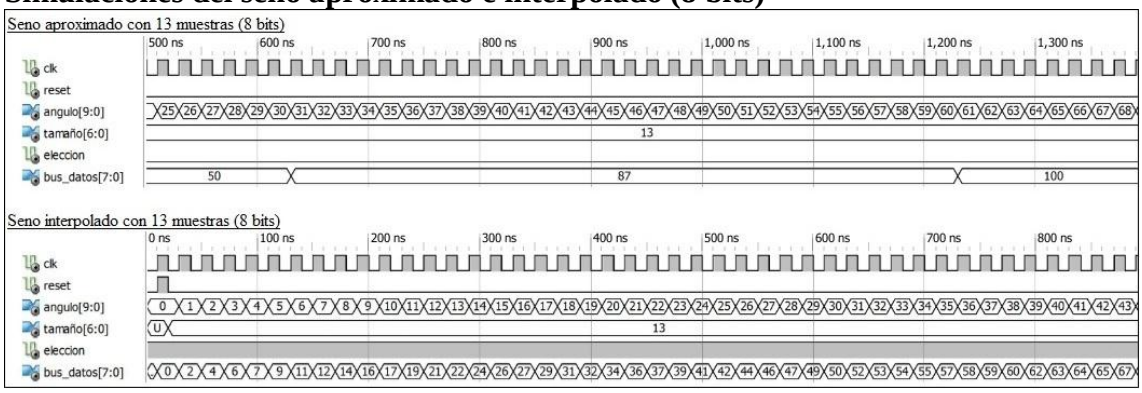
Banco de pruebas del diseño implementado en FPGA

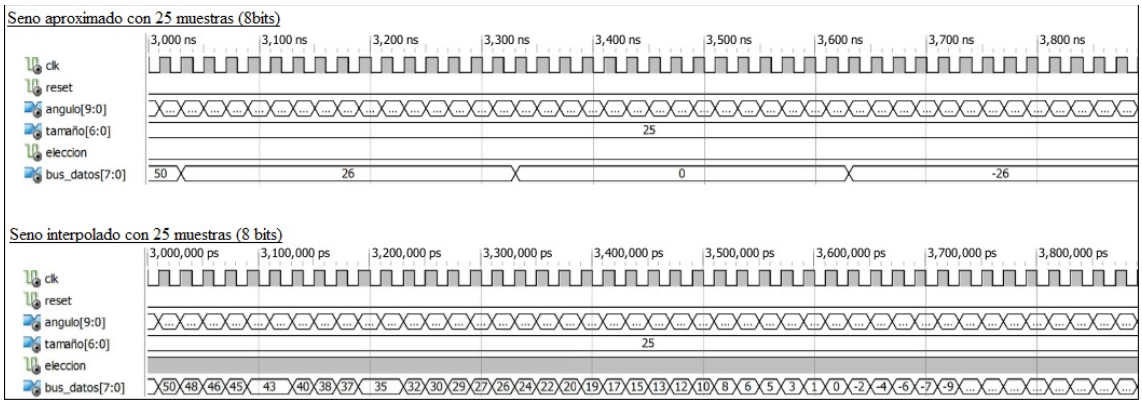
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_seno IS
END tb_seno;

ARCHITECTURE behavior OF tb_seno IS
 COMPONENT seno
  PORT(  clk        : IN  std_logic;
         reset      : IN  std_logic;
         int        : IN  std_logic;
         cap        : IN  std_logic;
         res        : IN  std_logic;
         s0         : IN  std_logic;
         s1         : IN  std_logic;
         s2         : IN  std_logic;
         s3         : IN  std_logic;
         led_ini    : OUT std_logic;
         led_1      : OUT std_logic;
         led_2      : OUT std_logic;
         led_3      : OUT std_logic;
         led_ang    : OUT std_logic;
         led_sen    : OUT std_logic;
         signo      : OUT std_logic;
         digito_7seg: OUT td_logic_vector(7 downto 0);
         digitos    : OUT std_logic_vector(3 downto 0));
 END COMPONENT;

   --Inputs
   signal clk   : std_logic := '0';
   signal reset : std_logic := '0';
   signal int   : std_logic := '0';
   signal cap   : std_logic := '0';
   signal res   : std_logic := '0';
   signal s0    : std_logic := '0';
   signal s1    : std_logic := '0';
   signal s2    : std_logic := '0';
   signal s3    : std_logic := '0';

 --Outputs
   signal led_ini     : std_logic;
   signal led_1       : std_logic;
   signal led_2       : std_logic;
   signal led_3       : std_logic;
   signal led_ang     : std_logic;
   signal led_sen     : std_logic;
   signal signo       : std_logic;
   signal digito_7seg : std_logic_vector(7 downto 0);
   signal digitos     : std_logic_vector(3 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)
   uut: seno PORT MAP (clk => clk, reset => reset, int => int, cap=> cap,
   res => res, s0 => s0, s1 => s1, => s2, s3 => s3, led_ini => led_ini,
   led_1 => led_1, led_2 => led_2, _3 => led_3, led_ang => led_ang,
   led_sen => led_sen, signo => signo, digito_7seg => digito_7seg,
   digitos => digitos);
```

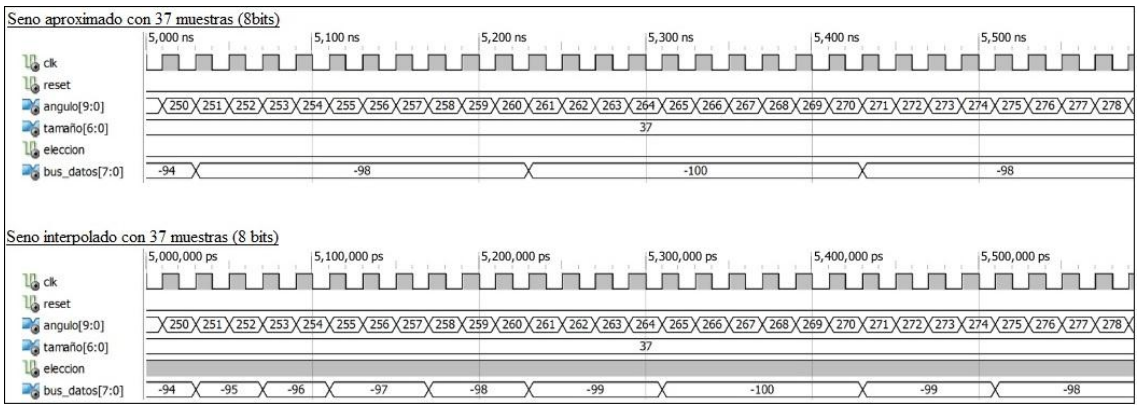
```
    -- Clock process definitions
    clk_process :process
    begin
              clk <= '0';
              wait for clk_period/2;
              clk <= '1';
              wait for clk_period/2;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
        reset <= '1';
        wait for 100 ns;
        reset <= '0';
        wait for 100 ns;
        int <= '1';
        wait for 100 ns;
        int <= '0';
        wait for 100 ns;
        s3 <= '0';   s2 <= '0';   s1 <= '0';   s0 <= '1';
        wait for 100 ns;
        cap <= '1';
        wait for 100 ns;
        cap <= '0';
        wait for 100 ns;
        s3 <= '0';   s2 <= '1';   s1 <= '1';   s0 <= '1';
        wait for 100 ns;
        cap <= '1';
        wait for 100 ns;
        cap <= '0';
        wait for 100 ns;
        s3 <= '0';   s2 <= '1';   s1 <= '0';   s0 <= '1';
        wait for 100 ns;
        cap <= '1';
        wait for 100 ns;
        cap <= '0';
        wait for 100 ns;
        cap <= '1';
        wait for 100 ns;
        cap <= '0';
        wait for 1000 ns;
        res <= '1';
        wait for 100 ns;
        res <= '0';

       wait;
    end process;
END;
```

Simulaciones de los diseños

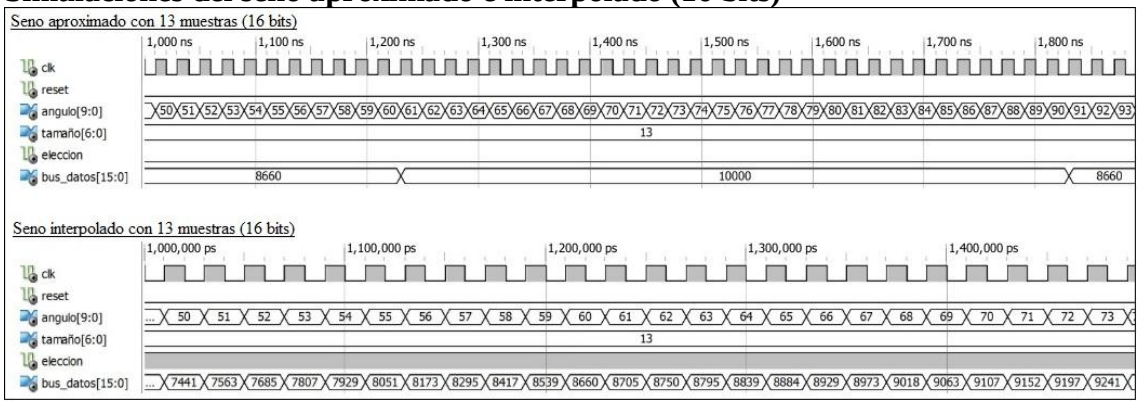
Simulaciones del seno aproximado e interpolado (8 bits)

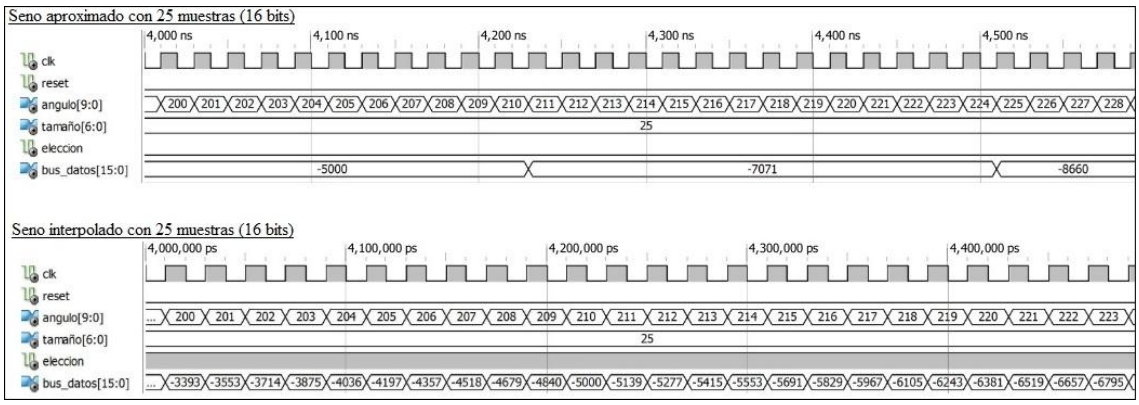


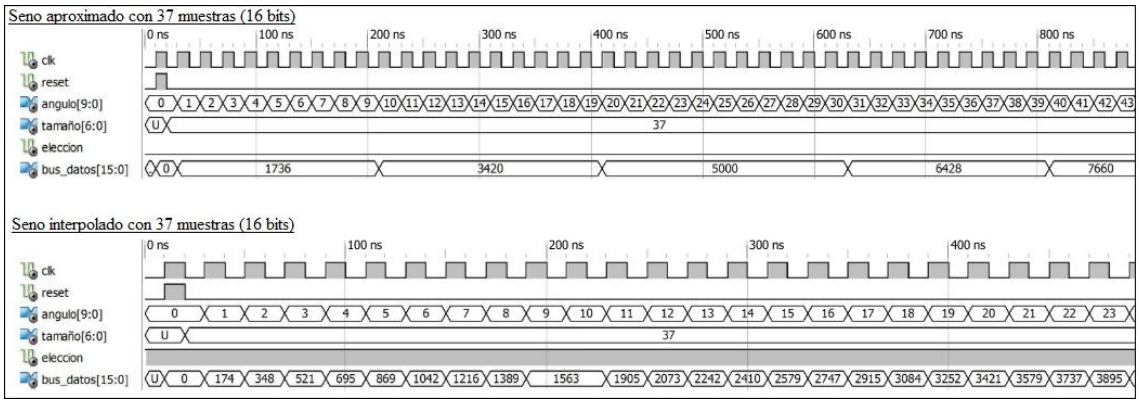




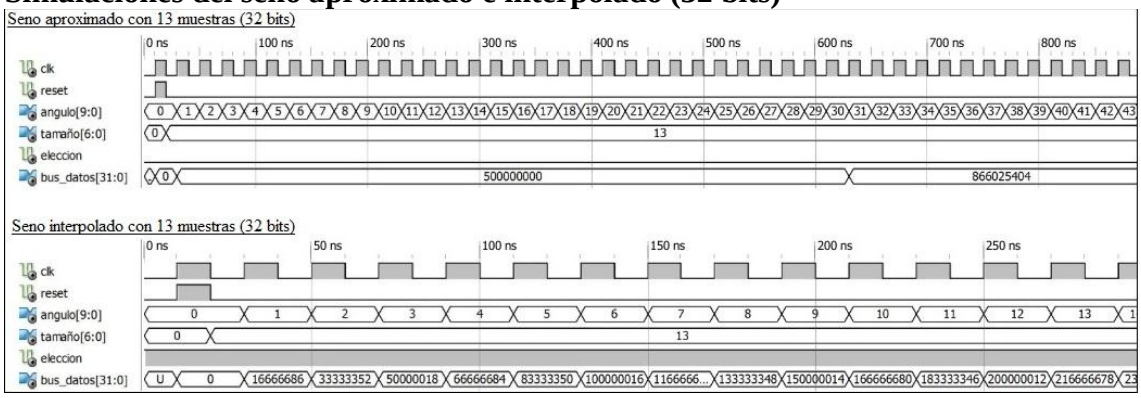
Simulaciones del seno aproximado e interpolado (16 bits)

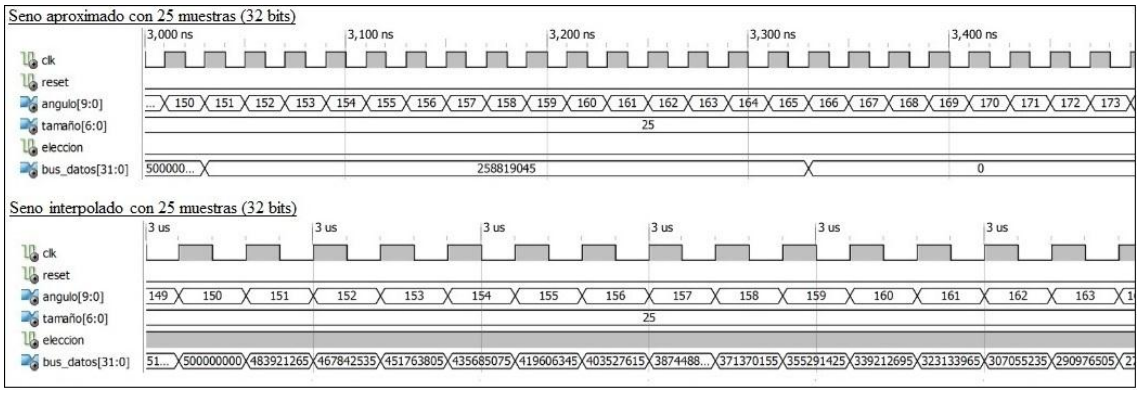


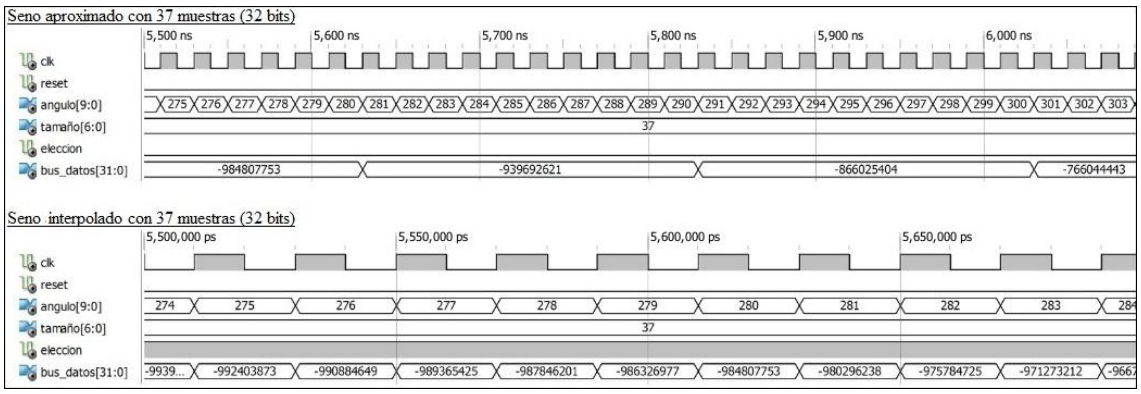




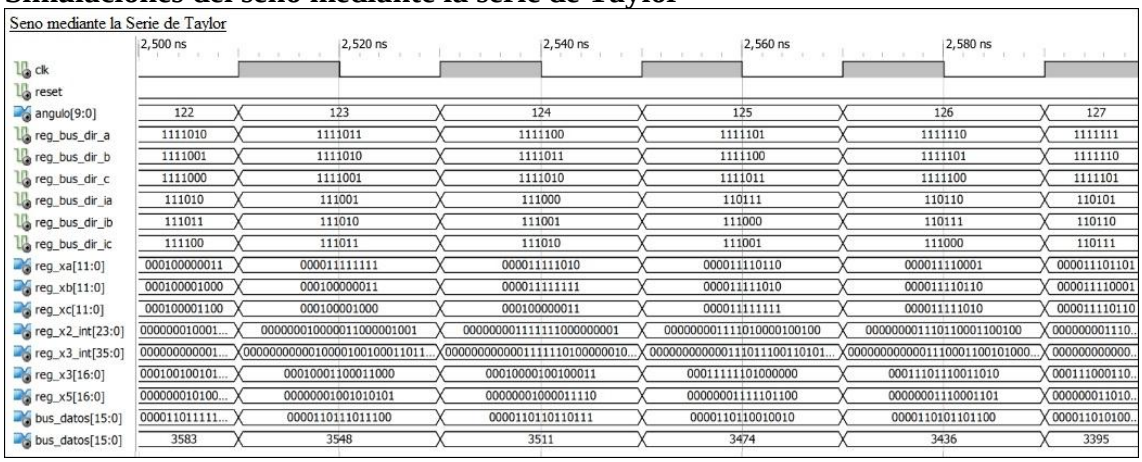
Simulaciones del seno aproximado e interpolado (32 bits)



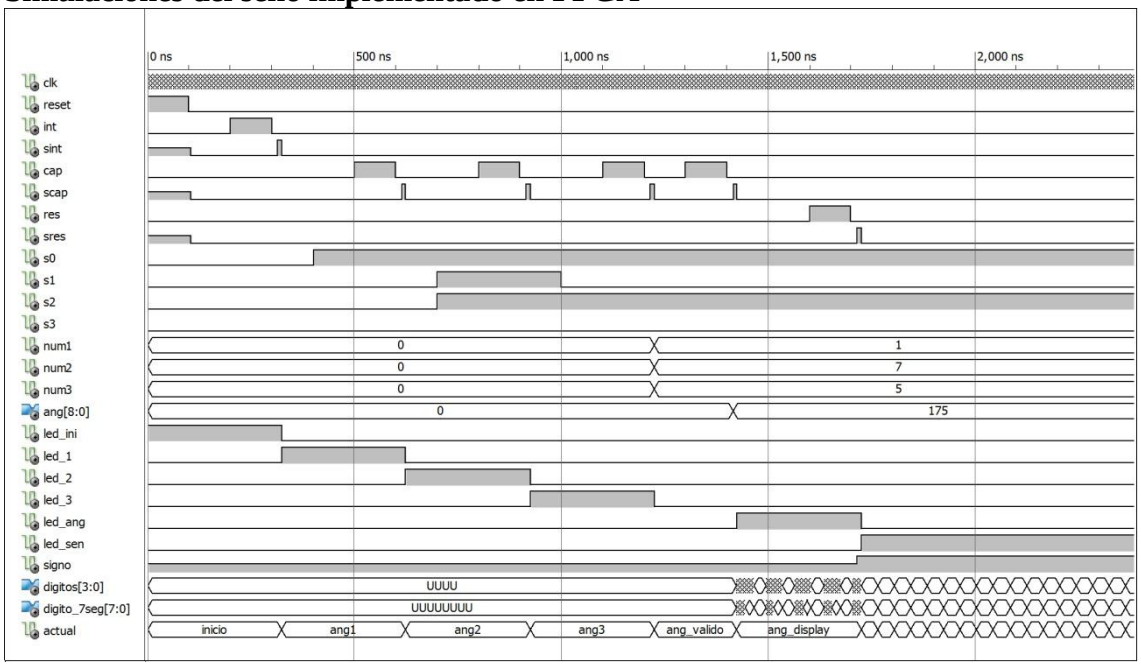


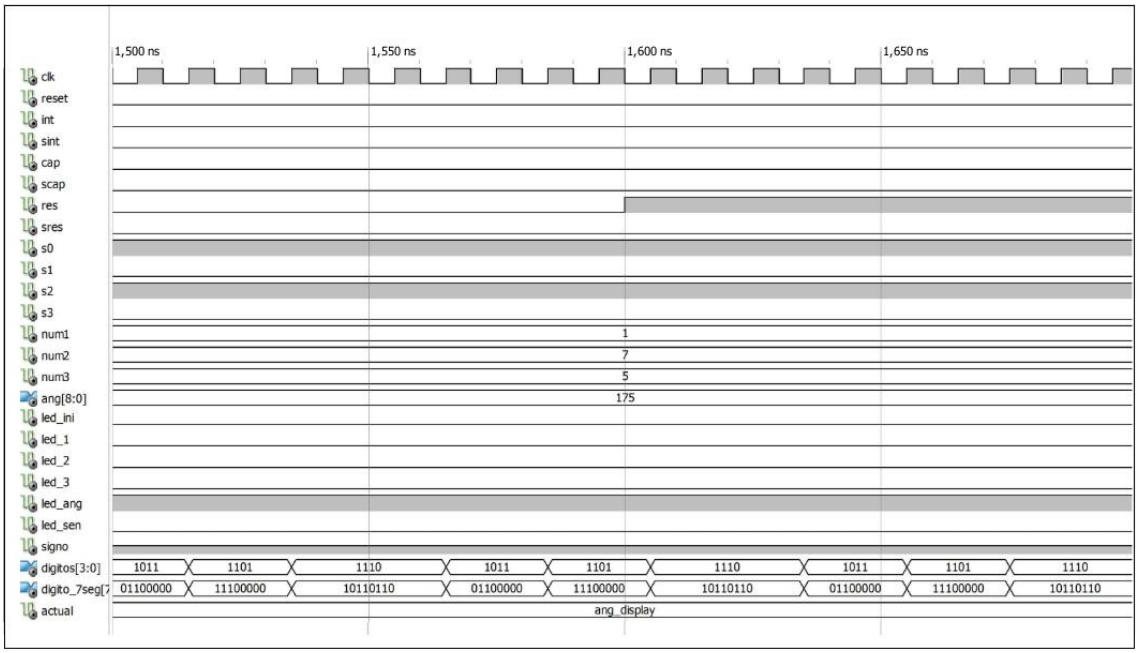


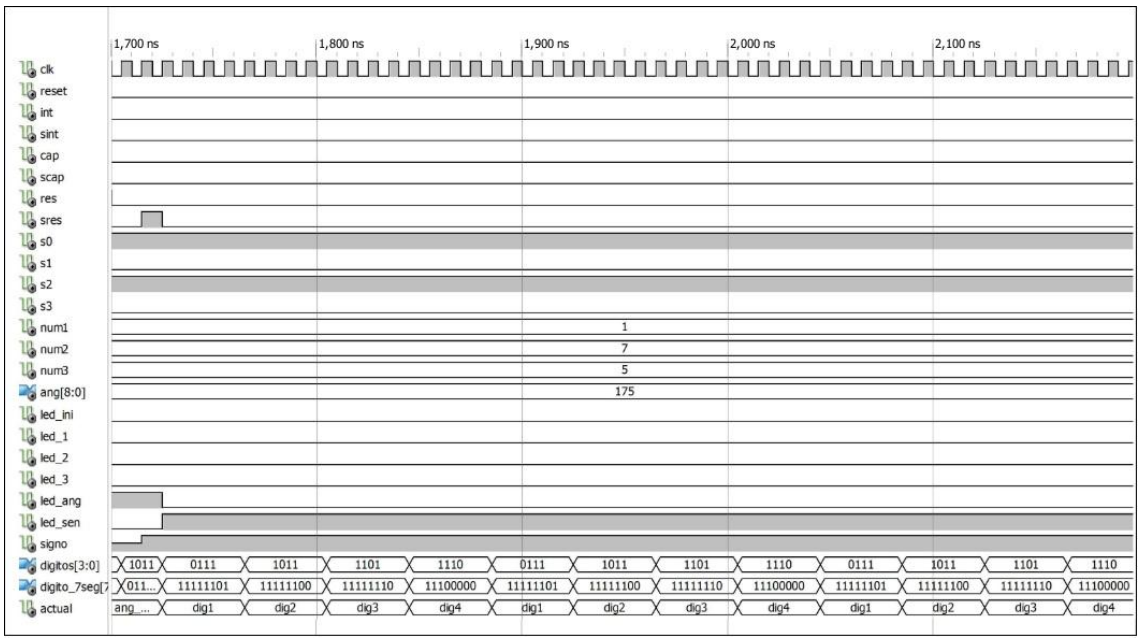
Simulaciones del seno mediante la serie de Taylor



Simulaciones del seno implementado en FPGA




1,500 ns
1,550 ns
1,600 ns
1,650 ns
clk
reset
int
sint
cap
scap
res
sres
s0
s1
s2
s3
num1
num2
num3
ang[8:0]
led_ini
led_1
led_2
led_3
led_ang
led_sen
signo
digitos[3:0]
digito_7seg[7
actual
1
7
5
175
1011 1101 1110 1011 1101 1110 1011 1101 1110
01100000 11100000 10110110 01100000 11100000 10110110 01100000 11100000 10110110
ang_display


1,700 ns
1,800 ns
1,900 ns
2,000 ns
2,100 ns
clk
reset
int
sint
cap
scap
res
sres
s0
s1
s2
s3
num1
num2
num3
ang[8:0]
led_ini
led_1
led_2
led_3
led_ang
led_sen
signo
digitos[3:0]
digito_7seg[7
actual
1
7
5
175
1011 0111 1011 1101 1110 0111 1011 1101 1110 0111 1011 1101 1110
011... 11111101 11111100 11111110 11100000 11111101 11111100 11111110 11100000 11111101 11111100 11111110 11100000
ang_... dig1 dig2 dig3 dig4 dig1 dig2 dig3 dig4 dig1 dig2 dig3 dig4

Código en MATLAB para el cálculo de errores

Error entre aproximar o interpolar un seno con 13, 25 o 37 muestras (8 bits)

```
fid_int=fopen('vhdl_outfile_13_interpolacion.txt','r');  % 13,25,37
fid_apr=fopen('vhdl_outfile_13_aproximacion.txt','r');   % 13,25,37
[A,cont_A]=fscanf(fid_int,'%d');
[B,cont_B]=fscanf(fid_apr,'%d');
fod=fopen('matlab_aprox-inter_13.txt','w');

for i=1:1:361
resultados(i)=abs(((0.01*A(i)-0.01*B(i))/(0.01*A(i)))*100);
fprintf(fod,'%.2f\n', resultados(i));
j(i)=i;
end

figure
x = 0:1:360;
y1 = B/100;
y2 = resultados;
[AX,H1,H2] = plotyy(x,y1,x,y2,'plot');
set(get(AX(1),'Ylabel'),'String','Seno (radianes)')
set(get(AX(2),'Ylabel'),'String','Error (%)')
xlabel('Ángulos (°)')
title('Resultados con 13 muestras')

hold on
z = 0:1:360;
f = A/100;
plot(z,f,'--c')
legend('Seno aproximado','Seno interpolado','Error')
```

Error entre aproximar o interpolar un seno con 13, 25 o 37 muestras (16 bits)

```
fid_int=fopen('vhdl_outfile_13_interpolacion.txt','r');  % 13,25,37
fid_apr=fopen('vhdl_outfile_13_aproximacion.txt','r');   % 13,25,37
[A,cont_A]=fscanf(fid_int,'%d');
[B,cont_B]=fscanf(fid_apr,'%d');
fod=fopen('matlab_aprox-inter_13.txt','w');

for i=1:1:361
resultados(i)=abs(((0.0001*A(i)-0.0001*B(i))/(0.0001*A(i)))*100);
fprintf(fod,'%.2f\n', resultados(i));
end

figure
x = 0:1:360;
y1 = B/10000;
y2 = resultados;
[AX,H1,H2] = plotyy(x,y1,x,y2,'plot');
set(get(AX(1),'Ylabel'),'String','Seno (radianes)')
set(get(AX(2),'Ylabel'),'String','Error (%)')
xlabel('Ángulos (°)')
title('Resultados con 13 muestras')  %13.25.37 muestras

hold on
z = 0:1:360;
f = A/10000
plot(z,f,'--c')
legend('Seno aproximado','Seno interpolado','Error')
```

Error entre aproximar o interpolar un seno con 13, 25 o 37 muestras (32 bits)

```
fid_int=fopen('vhdl_outfile_13_interpolacion.txt','r');  % 13,25,37
fid_apr=fopen('vhdl_outfile_13_aproximacion.txt','r');   % 13,25,37
[A,cont_A]=fscanf(fid_int,'%d');
[B,cont_B]=fscanf(fid_apr,'%d');
fod=fopen('matlab_aprox-inter_13.txt','w');

for i=1:1:361
resultados(i)=abs((0.000000001*A(i)-0.000000001*B(i)) /(0.000000001*A(i)));
resultados(i)=resultados(i)*100;
fprintf(fod,'%.2f\n', resultados(i));
end

figure
x = 0:1:360;
y1 = B/1000000000;
y2 = resultados;
[AX,H1,H2] = plotyy(x,y1,x,y2,'plot');
set(get(AX(1),'Ylabel'),'String','Seno (radianes)')
set(get(AX(2),'Ylabel'),'String','Error (%)')
xlabel('Ángulos (°)')
title('Resultados con 13 muestras')  %13.25.37 muestras

hold on
z = 0:1:360;
f = A/1000000000
plot(z,f,'--c')
legend('Seno aproximado','Seno interpolado','Error')
```

Error entre un seno interpolado (13, 25 o 37 muestras) y un seno de matlab (8 bits)

```
fid=fopen('vhdl_outfile_13_interpolacion.txt','r');   % 13,25,37
[A,cont]=fscanf(fid,'%d');
fod=fopen('matlab_inter-sin_13.txt','w');

j=0;
for i=1:1:361
resultados(i)=abs(((sin(j*2*pi/360)-((A(i))/100))/(sin(j*2*pi/360)))*100);
fprintf(fod,'%.2f\n', resultados(i));
j=j+1;
end

figure
x = 0:1:360;
y1 = A/100;
y2 = resultados;
[AX,H1,H2] = plotyy(x,y1,x,y2,'plot');
set(get(AX(1),'Ylabel'),'String','Seno (radianes)')
set(get(AX(2),'Ylabel'),'String','Error (%)')
xlabel('Ángulos (°)')
title('Resultados con 13 muestras')

hold on
z = 0:1:360;
f = sin(z*2*pi/360)
plot(z,f,'--c')
legend('Seno interpolado','Seno matlab','Error')
```

Error entre un seno interpolado (13, 25 o 37 muestras) y un seno matlab (16 bits)

```
fid=fopen('vhdl_outfile_13_interpolacion.txt','r');   % 13,25,37
[A,cont]=fscanf(fid,'%d');
fod=fopen('matlab_inter-sin_13.txt','w');

j=0;
for i=1:1:361
resultados(i)=abs((sin(j*2*pi/360)-((A(i))/10000))/(sin(j*2*pi/360)));
resultados(i)=resultados(i)*100;
fprintf(fod,'%.2f\n', resultados(i));
j=j+1;
end

figure
x = 0:1:360;
y1 = A/10000;
y2 = resultados;
[AX,H1,H2] = plotyy(x,y1,x,y2,'plot');
set(get(AX(1),'Ylabel'),'String','Seno (radianes)')
set(get(AX(2),'Ylabel'),'String','Error (%)')
xlabel('Ángulos (°)')
title('Resultados con 13 muestras')

hold on
z = 0:1:360;
f = sin(z*2*pi/360)
plot(z,f,'--c')
legend('Seno interpolado','Seno (matlab)','Error')
```

Error entre un seno interpolado (13, 25 o 37 muestras) y un seno matlab (32 bits)

```
fid=fopen('vhdl_outfile_13_interpolacion.txt','r');   % 13,25,37
[A,cont]=fscanf(fid,'%d');
fod=fopen('matlab_inter-sin_13.txt','w');

j=0;
for i=1:1:361
resultados(i)=abs((sin(j*2*pi/360)-((A(i))/1000000000))/(sin(j*2*pi/360)));
resultados(i)=resultados(i)*100;
fprintf(fod,'%.2f\n', resultados(i));
j=j+1;
end

figure
x = 0:1:360;
y1 = A/1000000000;
y2 = resultados;
[AX,H1,H2] = plotyy(x,y1,x,y2,'plot');
set(get(AX(1),'Ylabel'),'String','Seno (radianes)')
set(get(AX(2),'Ylabel'),'String','Error (%)')
xlabel('Ángulos (°)')
title('Resultados con 13 muestras')

hold on
z = 0:1:360;
f = sin(z*2*pi/360)
plot(z,f,'--c')
legend('Seno interpolado','Seno matlab','Error')
```

Error entre un seno mediante la serie de Taylor y un seno de matlab

```
fid=fopen('taylor_vhdl.txt','r');
[A,cont]=fscanf(fid,'%d');
fod=fopen('matlab_taylor-sin.txt','w');

j=0;
for i=1:1:361
resultados(i)=abs(((sin(j*2*pi/360)-((A(i))/4096))/(sin(j*2*pi/360)))*100);
fprintf(fod,'%.4f\n', resultados(i));
j=j+1;
end

x = 0:1:360;
y1 = A/4096;
y2 = resultados;
[AX,H1,H2] = plotyy(x,y1,x,y2,'plot');
set(get(AX(1),'Ylabel'),'String','Seno (radianes)')
set(get(AX(2),'Ylabel'),'String','Error (%)')
xlabel('Ángulos (°)')
title('Resultados con la serie de Taylor')

hold on
z = 0:1:360;
f = sin(z*2*pi/360)
plot(z,f,'--c')
legend('Seno taylor','Seno matlab','Error')
```

Glosario

FPGA: *Field Programmable Gate Array.*
ASIC: *Application Specific Integration.*
PAL: *Programmable Logic Device.*
CPLD: *Complex Programable Logic.*
PC: *Personal Computer.*
VHSIC: *Very High Speed Integrated Circuit.*
HDL: *Hardware Description Language.*
VHDL: *Very High Speed Integrated Circuit Hardware Description Language.*
LUT: *Look Up Table.*
ROM: *Read-Only Memory.*
RAM: *Random-Acces Memory.*
CAD: *Computer-Aided Design.*
PSPICE: *Simulation Program with Integrated Circuit Emphasis.*
RTL: *Resistor-Transistor Logic.*
MOS: *Metal Oxide Semiconductor.*
CMOS: *Complementary Metal Oxide Semiconductor.*
NMOS: *Negative-type Metal-Oxide Semiconductor.*
PMOS: *Positive-type Metal-Oxide Semiconductor.*

Bibliografía

[1] Beatriz Brogueras García, 'Diseño y Validación del Control Digital de un Inversor de Potencia en Ejes de Referencia Síncronos Conectado a Red'. Proyecto Fin de carrera, Universidad Carlos III de Madrid, 2011.

[2] http://www.ni.com/fpga/esa/. Beneficios del uso de FPGAS. Junio 2013.

[3] Javier Carroquino Cañas, 'Trigonometría. Razones Trigonométricas'. Junio 2010.

[4] http://www.esacademic.com/dic.nsf/eswiki/265988. Imágen de las relaciones trigonométricas más significativas. Junio 2013.

[5] Ron Larson, Robert P. Hostetler, Bruce H. Edwards, 'Cálculo'. McGraw-Hill, 2006.

[6] http://www.matematicasvisuales.com/html/analisis/taylor/sinTaylor.html. Representación gráfica del seno mediante la serie de Taylor. Junio 2013.

[7] Villar, E., Terés, L., Olcoz, S., Torroja, Y., 'VHDL Lenguaje estándar de diseño electrónico'. McGraw-Hill, 1998.

[8] Pardo, F. y Boluda J., "VHDL Lenguaje para síntesis y modelado de circuitos," Alfaomega Grupo Editor, 2000.

[9] Luis Entrena. 'Introducción al Diseño de Circuitos Digitales con VHDL'. Universidad Carlos III de Madrid, 2004.

[10] Ferrero, F.J. 'Descripción de circuitos digitales mediante VHDL'. Universidad de Oviedo. Mayo 1999.

[11] 'IEEE Standard VHDL Language Reference Manual'. Published by the Institute of Electrical and Electronics Engineers, 1994.

[12] IEEE Standard VHDL Language Reference Manual, Std. 1076, 2000 Edition.

[13] http://centrodeartigos.com/articulos-informativos/article_69419.html. Historia de la empresa Xilinx. Junio 2013.

[14] ISE In-Depth Tutorial www.xilinx.com UG695 (v 12.3). Septiembre 2010.

[15] ISE Design Suite Software Manuals and Help UG681 (v 11.4). Diciembre 2009.

[16] Leopoldo Silva Bijit, 'Codificación binaria'. 2010.

[17] Thomas L. Floyd, 'Fundamentos de Sistemas digitales'. Prentice Hall. 7ª edicion.

[18] http://es.ncalculators.com/digital-computation/calculadora.htm. Calculadora binaria online. Junio 2013.

[19] Germán León Navarro. Multiplicador Segmentado. Arquitectura de computadores. Universidad Jaume-I.

[20] ISim User Guide UG660 (v 11.3). Septiembre 2009.

[21] Mª Cristina Casado Fernández, 'Manual Básico de Matlab'. Servicios Informáticos UCM, Apoyo a Investigación y Docencia.

[22] Spartan-3E Starter Kit Board User Guide. UG230 (v1.0) March 9, 2006.

[23] Data Sheet Seven Segment Display SA52-11EWA/GWA/YWA/SRWA.

[24] Data Sheet LedTech Four Digit GaP HER Seven Segment Display LM5644R-11N.